April 1982

Interfacing Dynamic RAM to iAPX 86, 88 Systems Using the Intel 8202A and 8203

Brad May
Peripheral Component Applications Engineering

ORDER NUMBER: 210398-001

INTRODUCTION

The designer of a microprocessor-based system has two basic types of devices available to implement a random access read/write memory — static or dynamic RAM. Dynamic RAMs offer many advantages. First, dynamic RAMs have four times the density (number of bits per device) of static RAMs, and are packaged in a 16-pin DIP package, as opposed to the 20-pin or larger DIPs used by static RAMs; this allows four times as many bytes of memory to be put on a board, or alternatively, a given amount of memory takes much less board space. Second, the cost per bit of dynamic RAMs is roughly one-fourth that of statics. Third, static RAMs use about one-sixth the power of static RAMs, so power supplies may be smaller and less expensive. These advantages are summarized in Table 1.

On the other hand, dynamic RAMS require complex support functions which static RAMs don't, including

- address multiplexing
- timing of addresses and control strobes
- refreshing, to prevent loss of data
- arbitration, to decide when refresh cycles will be performed.

Table 1. Comparison of Intel Static and Dynamic RAMs Introduced during 1981

	2164-15 (Dynamic)	2167-70 (Static)
Density (No. of bits)	64K	16K
No. of pins	16	20
Access time (ns)	150	70
Cycle time (ns)	300	70
Active power (ma)	60	125
Standby power (ma)	5	40
Approx. cost per bit (millicents/bit)	45	250

In addition, dynamic RAMs may not always be able to transfer data as fast as high-performance microprocessors require; wait states must be generated in this case. The circuitry required to perform these functions takes up board space, costs money, and consumes power, and so detracts from the advantages that make dynamic RAMs so appealing. Obviously, the amount of support circuitry should be minimized.

The Intel 8202A and 8203 are LSI dynamic RAM controller components. Either of these 40-pin devices alone does all of the support functions required by dynamic RAMs. This results in a minimum of board space, cost, and power consumption, allowing maximum advantage from the use of dynamic RAMs.

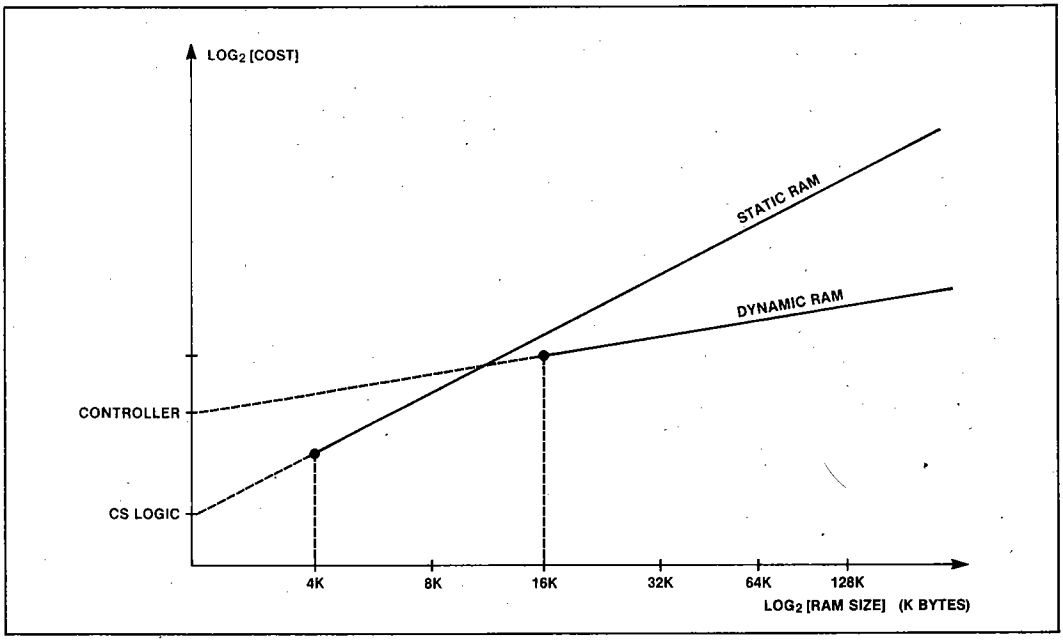


Figure 1. Implemented Cost of Static vs. Dynamic RAM

Figure 1 shows the relative cost of static and dynamic RAM, including support circuitry, as a function of memory size, using the Intel 8202A or 8203. For any memory larger than 16KBytes, the dynamic RAM is less expensive. Since the cost of the dynamic RAM controller is relatively independent of memory size, the cost advantge for dynamic RAM increases with increasing memory size.

This Application Note will describe the techniques of interfacing a dynamic RAM memory to an iAPX-86 or iAPX-88 system using either the 8202A or 8203 dynamic RAM controller. Various configurations of the 8086 and 8088 microprocessors, and those timings which they satisfy, are described. The Note concludes with examples of particular system implementations.

DYNAMIC RAMS

This section gives a brief introduction to the interfacing requirements for Dynamic RAMs. Later sections will describe the operation of the Intel 8202A and 8203 Dynamic RAM Controllers.

Device Description

The pinout of two popular families of dynamic RAMs, the Intel 2118 and 2164A, are shown in Figure 2. The 2118 is a 16,384 word by 1-bit dynamic MOS RAM. The 2164 is a 65,536 word by 1-bit dynamic MOS RAM. Both parts operate from a single +5v supply with a ± 10% tolerance, and both use the industry standard 16-lead pinout.

The two parts are pinout-compatible with the exception of the 2164 having one extra address input (A_7, pin 9); this pin is a no-connect in the 2118. Both parts are also compatible with the next generation of 256K dynamic RAMs (262,144 word by 1-bit), which will use pin 1 (presently a no-connect on both the 2118 and 2164A) for the required one extra address input (A_8). This makes it possible to use a single printed circuit board layout with any of these three types of RAM.

Addressing

Each bit of a dynamic RAM is individually addressable. Thus, a 2164A, which contains 2^{16} (or 65,536) bits of information, requires 16-bit addresses; similarly, the 2118, which contains 2^{14} (or 16,384) bits, requires 14-bit addresses.

In order to reduce the number of address pins required (and thus reduce device cost), dynamic RAMs time-multiplex addresses in two halves over the same pins. Thus a 2164A needs only 8 address pins to receive 16-bit addresses, and the 2118 needs only 7 for its 14-bit addresses. The first address is called the *row* address, and the second is called the *column* address. The row address is latched internal to the RAM by the falling edge of the $\overline{RAS}$ (*R*ow *A*ddress *S*trobe) control input; the column address is latched by the falling edge of the $\overline{CAS}$ (*C*olumn *A*ddress *S*trobe) control input. This operation is illustrated in Figure 3.

Dynamic RAMS may be visuallized as a two-dimensional array of single-bit storage cells arranged across the surface of the RAM's die. In the case of the 2164A, this array would consist of 2^8 (or 256) rows and 2^8 (or 256) columns, for a total of 2^{16} (or 65,526) total bit cells (Figure 4). This is the source of the "row address" and "column address" terminology. Bear in mind that any given RAM may not be physically implemented as described here; for instance, the 2164A actually contains four arrays, each one 2^7 rows by 2^7 columns.

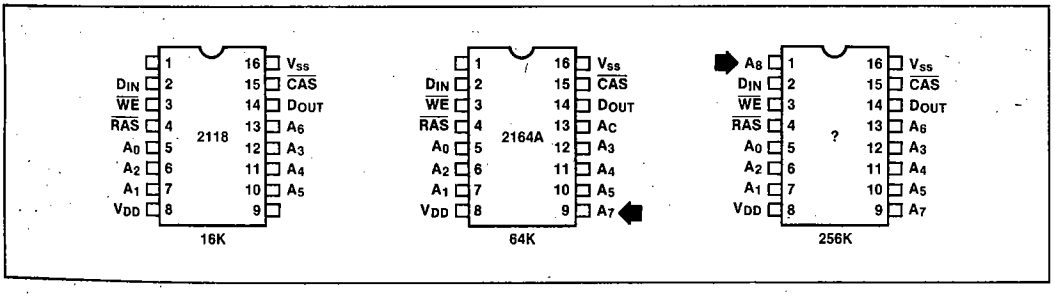


Figure 2. Dynamic RAM Pinout Compatibility

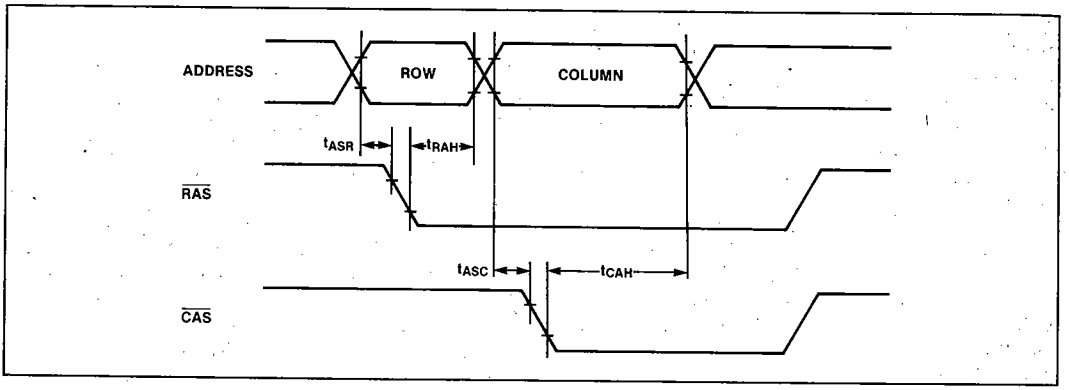


Figure 3. Dynamic RAM Addressing

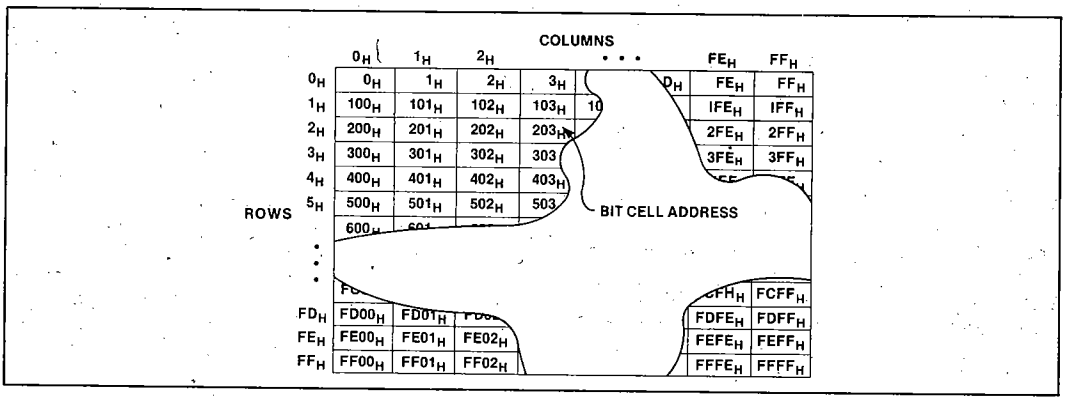


Figure 4. Bit Cell "Array"

Memory Cycles

In this Application Note, we will discuss three types of memory cycles — read, write, and $\overline{RAS}$-only refresh. Dymanic RAMs may perform other types of cycles as well; these are described in the dynamic RAM's data sheet.

Whether data is read or written during a memory cycle is determined by the RAM's $\overline{WE}$ control input. Data is written only when $\overline{WE}$ is active.

During a read cycle, the $\overline{CAS}$ input has a second function, other than latching the column address. $\overline{CAS}$ also enables the RAM data output (pin 14) when active, assuming $\overline{RAS}$ is also active. Otherwise, the data output is 3-stated. This allows multiple dynamic RAMs to have their data outputs tied in common.

During write cycles, data on the RAM data input pin is latched internally to the RAM by the falling edge of $\overline{CAS}$ or $\overline{WE}$, *whichever occurs last*. If $\overline{WE}$ goes active before $\overline{CAS}$ (the usual case, called an "early write"), write data is latched by the falling edge of $\overline{CAS}$. If $\overline{WE}$ goes active after $\overline{CAS}$ (called a "late write"), data is latched by the falling edge of $\overline{WE}$ (see Figure 5).

Late writes are useful in some systems where it is desired to start the memory cycle as quickly as possible, to maximize performance, but the CPU cannot get the write data to the dynamic RAMs quickly enough to be latched by $\overline{CAS}$. By delaying $\overline{WE}$, more time is allowed for write data to arrive at the dynamic RAMs.

Note that when "late write" is performed, $\overline{CAS}$ goes active while $\overline{WE}$ is still inactive; this indicates a read cycle, so the RAM enables its data output. So, if "late write" cycles are performed by a system, the RAM data inputs and data outputs must be electically isolated from each other to prevent contention. If no "late writes" are performed, the RAM data inputs and data outputs may be tied together at the RAM to reduce the number of board traces.

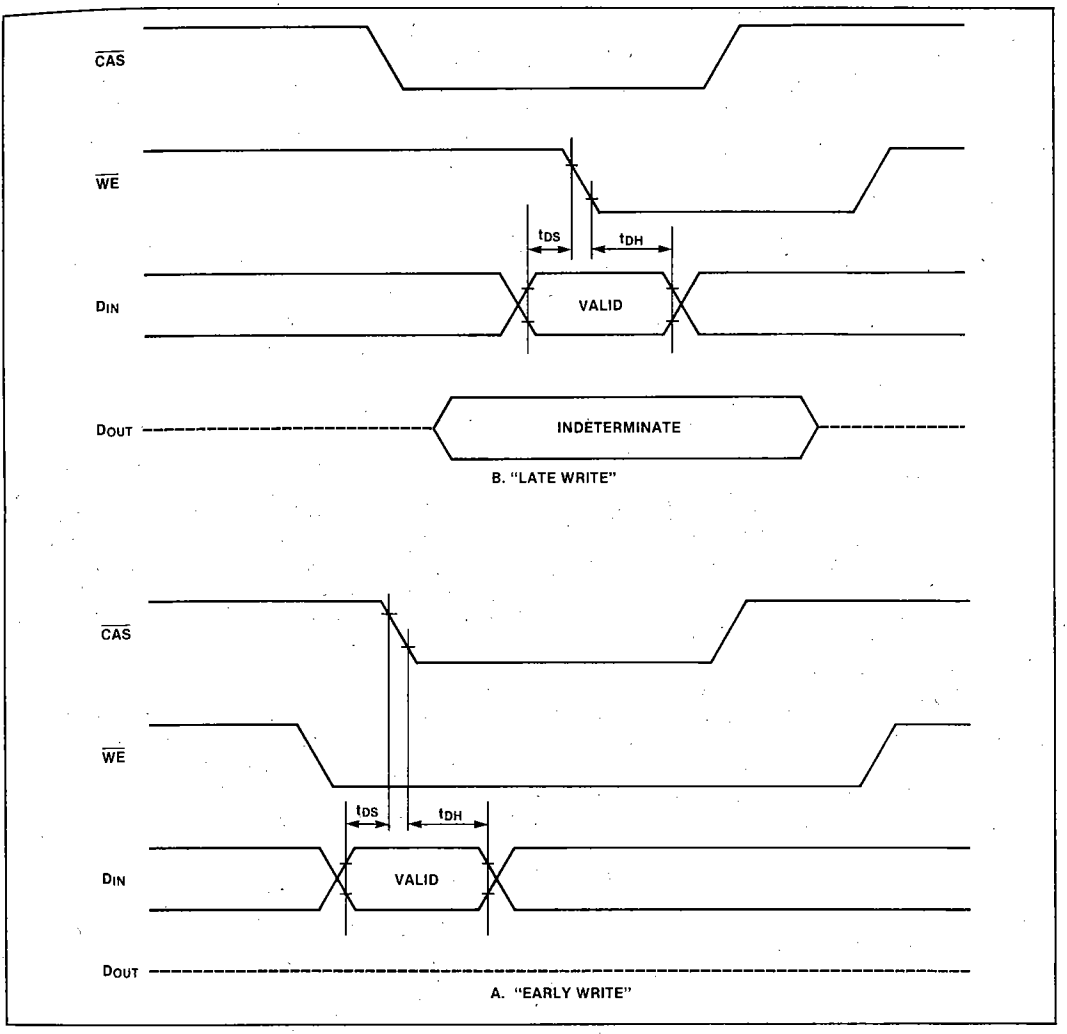


Figure 5. Dynamic RAM Write Cycles

Access Times

Each dynamic RAM has two different access times quoted for it — access time from $\overline{RAS}$ active (t_{RAC}) and access time from $\overline{CAS}$ active (t_{CAC}); these are illustrated in Figure 6. How do you know which to use? This depends on the timings of your RAM controller. First, the worst case delay from the memory read command active to $\overline{RAS}$ active (t_{CR}) and $\overline{CAS}$ active (t_{CC}) must be determined. Then the read data access time is the larger of the t_{CR}(Controller) + t_{RAC}(RAM) or t_{CC}(Controller) + t_{CAC}(RAM). An alternative way to determine whether to use t_{RAC} or t_{CAC} is to look at the dynamic RAM parameter for $\overline{RAS}$ active to $\overline{CAS}$ active delay, t_{RCD}. t_{RCD}*max* is a calculated value, and is shown on dynamic RAM data sheets as a reference point only. If the delay from $\overline{RAS}$ to $\overline{CAS}$ is less than or equal to t_{RCD}max, then t_{RAC} is the limiting access time parameter; if, on the other hand, the delay from $\overline{RAS}$ to $\overline{CAS}$ is greater than t_{RCD}max, then t_{CAC} is the limiting parameter. t_{RCD}max is not an operating limit, and this spec may be exceeded without affecting operation of the RAM. t_{RCD}*min*, on the other hand, is an operating limit, and the RAM will not operate properly if this spec is violated.

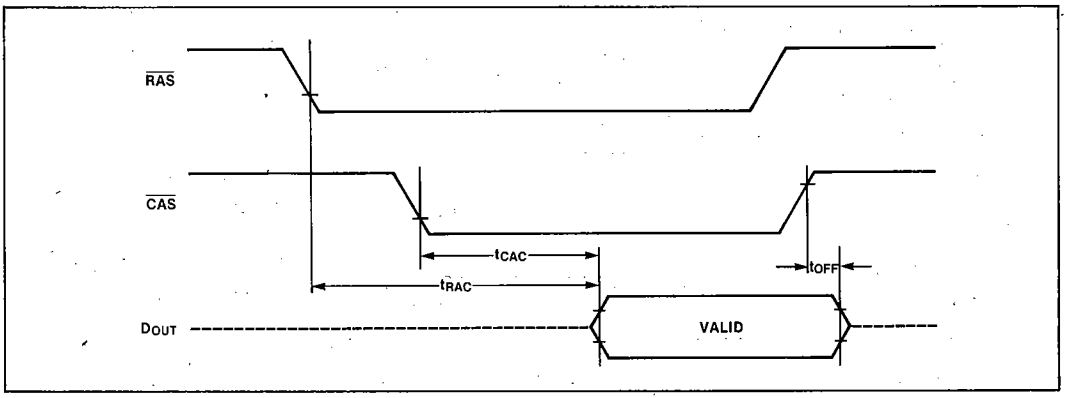


Figure 6. Dynamic RAM Access Times

Refresh

One unique requirement of dynamic RAMs is that they be *refreshed* in order to retain data. To see why this is so, we must look briefly at how a dynamic RAM is implemented.

Dynamic RAMs achieve their high density and low cost mostly because of the very simple bit-storage cell they use, which consists only of one transistor and a capacitor. The capacitor stores one bit as the presence (or absence) of charge. This capacitor is selectively accessed for reading and writing by enabling its associated transistor (see Figure 7).

Unfortunately, if left for very long, the charge will leak out of the capacitor, and the data will be lost. To prevent this, each bit-cell must be periodically read, the charge on the capacitor amplified, and the capacitor recharged to its initial state. The circuitry which does this amplification of charge is called a "sense amp". This must be done for every bit-cell every 2 ms or less to prevent loss of data.

Each column in a dynamic RAM has its own sense amp, so refresh can be performed on an entire row at a time. Thus, for the 2118, it is only necessary to refresh each of its 128 rows every 2 ms. Each row must be addressed via the RAM's address inputs to be refreshed. To simplify

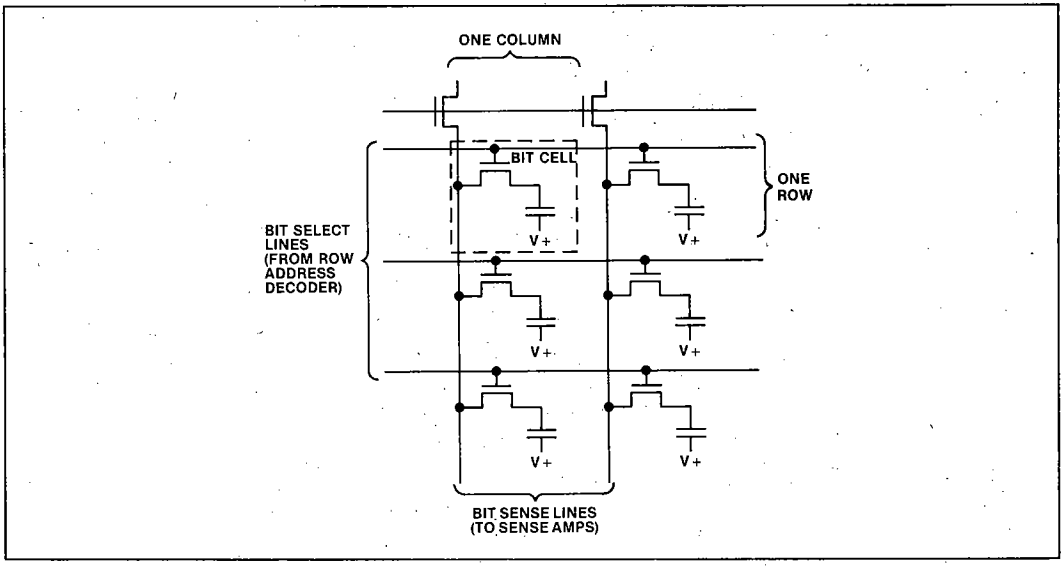


Figure 7. Dynamic RAM Cell

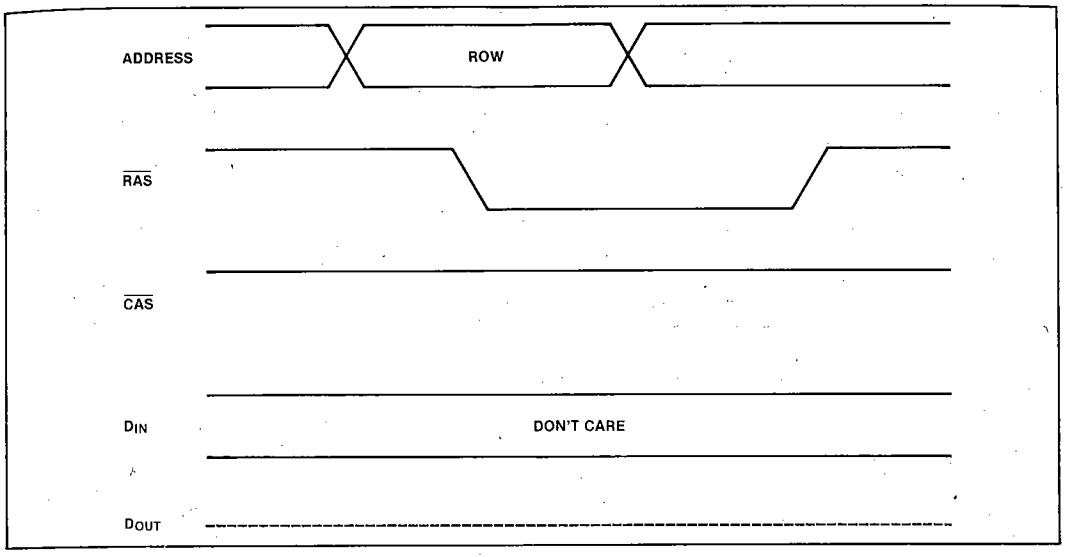


Figure 8. RAS-only Refresh

refresh, the 2164A is implemented in such a way that its refresh requirements are identical to the 2118; 128 rows every 2 ms. Some other 64K RAMs require 256 row refresh every 4 ms.

Refresh can be performed by a special cycle called a *RAS-only refresh*, shown in Figure 8. Only a row address is sent; that row is refreshed. No column address is sent, and no data is read or written during this cycle. Intel dynamic RAM controllers use this technique.

Any read, write, or read-modify-write cycle also refreshes the row addressed. This fact may be used to refresh the dynamic RAM without doing any special refresh cycles. Unfortunately, in general you cannot be sure that every row of every dynamic RAM in a system will be read from or written to every 2 ms, so refresh cannot be guaranteed by this method alone, except in special applications.

A third technique for refresh is called *hidden refresh*. This method is not popular in microprocessor systems, so it is not described here, but more information is available in the dynamic RAM's data sheet.

Three techniques for timing when refresh cycles are performed are in common use: burst refresh, distributed refresh, and transparent refresh.

Burst refresh means waiting almost 2 ms from the last time refresh was performed, then refreshing the entire memory with a "burst" of 128 refresh cycles. This method has the inherent disadvantage that during the time refresh is being performed (more than 40 microseconds for 128 rows) no read or write cycles can be performed. This severely limits the worst case response time to interrupts and makes this approach unsuitable for many systems.

As long as every row of the RAM is refreshed every 2 ms, the distribution of individual refresh cycles is unimportant. *Distributed refresh* takes advantage of this fact by performing a single refresh cycle every 2 ms/128, or about every 15 microseconds. In this way, the refresh requirements of the RAM are satisfied, but the longest time that read and write cycles are delayed because of refresh is minimized. Those few dynamic RAMs which use 256 row refresh allow 4 ms for the refresh to be completed, so the distributed refresh period is still 15 microseconds.

The third technique is called *transparent* (or "hidden" or "syncronous") *refresh*. This takes advantage of the fact that many microprocessors wait a fixed length of time after fetching the first opcode of an instruction to decode it. This time is necessary to determine what to do next (i.e. fetch more opcode bytes, fetch operands, operate on internal registers, etc.); this time may be longer than the time required for a RAM refresh cycle. If the status outputs of the CPU can be examined to determine which memory cycles are opcode fetches, a refresh cycle may be performed immediately afterward (Figure 9). In this way, refresh cycles will never interfere with read or write cycles, and so appear "transparent" to the microprocessor.

Transparent refresh has the disadvantage that if the microprocessor ever stops fetching opcodes for very

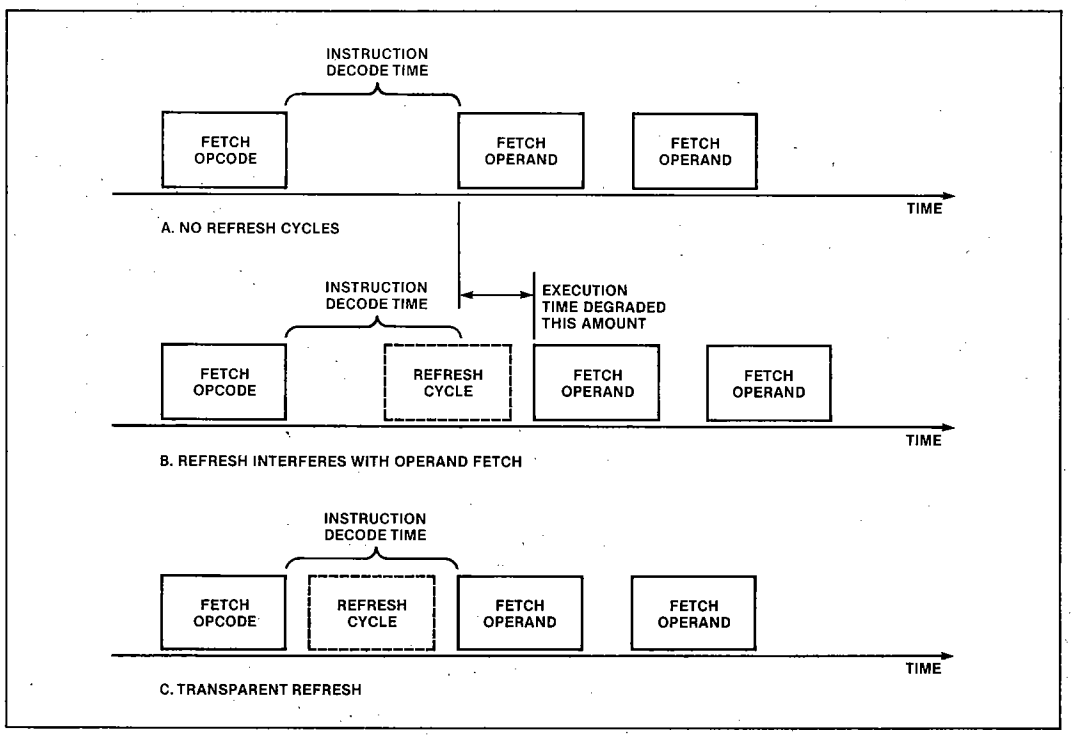


Figure 9. Transparent Refresh

long, due to a HOLD, extended DMA transfers, or when under hardware emulation, no refresh cycles will occur and RAM data will be lost. This puts restrictions on the system design. Also, high speed microprocessors do not allow sufficient time between opcode fetches and susequent bus cycles for a complete RAM refresh cycle to be performed, so they must wait for the refresh cycle to complete before they can do a subsequent bus cycle. These microprocessors cannot use transparent refresh to any advantage. Transparent refresh is useful for microprocessors like the Intel 8085 operating at low clock frequencies.

The 8086 and 8088, however, prefetch opcodes into a queue which is several bytes long. This prefetching is independent of the actual decoding and execution of the opcodes, and there is no time at which it can be guaranteed that the 8086 or 8088 will not request a memory cycle. So transparent refresh is not applicable to these microprocessors.

The 8202A and 8203 perform distributed and/or transparent refresh. Each device has an internal timer which automatically generates a distributed refresh cycle every 15.6 microseconds or less. In addition, an external refresh request input (REFRQ) allows the microprocessor's status to be decoded to generate a refresh cycle for transparent refresh. If, for whatever reason, no external REFRQ is generated for 15 microseconds, the internally generated refresh will take over, so memory integrity will be guaranteed.

Arbitration

Because RAMs cannot do a read or write cycle and a refresh cycle at the same time, some form of *arbitration* must be provided to determine when refresh cycles will be performed.

Arbitration may be done by the microprocessor or by the dynamic RAM controller. Microprocessor arbitration may be implemented as follows:

A counter, running from the microprocessor's clock, is used to time the period between refresh cycles. At terminal count, the arbitration logic asserts the bus request signal to prevent the microprocessor from performing any more memory cycles. When the microprocessor responds with a bus grant, the arbitration logic generates a refresh cycle (or cycles, if burst refresh is

used). After refresh is complete, the arbitration logic releases the bus. This method has several disadvantages: First, time is wasted in exchanging bus control, which would not be required if the RAM controller did arbitration. Second, while refresh is being performed, *all* bus activity is stopped; for instance, even if the microprocessor is executing out of ROM at the time, it must stop until refresh is over. Third, bursts of DMA transfers must be kept very short, as refresh cannot be performed while DMA is in progress.

Some microprocessors, such as the Zilog Z-80, generate refresh cycles themselves after instruction fetches. This removes the need for external arbitration logic, but still has several disadvantages: First, DMA bursts still must be kept short to allow the CPU to do refresh. Second, this method adds to the complexity of the microprocessor, without removing the need for the RAM controller which is still required to do address multiplexing and $\overline{RAS}$, $\overline{CAS}$ and $\overline{WE}$ timing. Microprocessor refresh can cause problems of RAM compatibility; for instance, the Z-80 only outputs a 7-bit refresh address, which means some 64K RAMs which use 256 row refresh cannot be used with the Z-80. Also, since the Z-80 refresh cycle is a fixed length (no wait states), faster speed selections of the Z-80 are not compatible with slower dynamic RAMs. Third, systems employing multiprocessing or DMA are harder to implement, because of the difficulty in insuring the microprocessor will be able to perform refresh.

It is preferable to have arbitration performed by the dynamic RAM controller itself. This method avoids all the problems described above, but introduces a complication. If the microprocessor issues a read or write command while the dynamic RAM is in the middle of a refresh cycle, the RAM controller must make the microprocessor wait until it is done with the refresh before it can complete the read or write cycle. This means that from when the microprocessor activates the read or write signal, the time until the cycle can be completed can vary over a range of roughly 200 to 700 ns. Because of this, an acknowledge signal from the dynamic RAM controller is required to tell the microprocessor the memory cycle it requested is complete. This signal goes to the microprocessor's READY logic.

Memory Organization

As each dynamic RAM operates on only one bit at a time, multiple RAMs must be operated in parallel to operate on a word at a time. RAMs operated in this way are called a *bank* of RAM. A bank consists of as many RAMs as there are bits in the memory word. When used in this way, all address and control lines are tied to all RAMs in the bank.

A single bank of RAM will provide 64K words of memory in the case of the 2164A, or 16K words in the case of the 2118. To provide more memory words, multiple banks of RAM are used. In this case, all address, CAS, and $\overline{WE}$ lines are tied to all RAMs, but each bank of RAM has *its own* $\overline{RAS}$. Each bank knows whether it is being addressed during a read or write operation by whether or not its $\overline{RAS}$ input was activated — if not, then all other inputs are ignored during that cycle.

Data outputs for RAMs in corresponding bit positions in each of the banks may be tied in common, since they are 3-state outputs; even though $\overline{CAS}$ is connected to all banks of RAM, only that bank whose $\overline{RAS}$ is active will enable its data outputs in response to $\overline{CAS}$ going active. Data inputs for RAMs in corresponding bit positions in each of the banks are also tied in common.

INTEL DYNAMIC RAM CONTROLLERS

The Intel 8202A and 8203 Dynamic RAM Controllers each provide all the interface logic needed to use dynamic RAMs in microprocessor systems, in a single chip. Either the 8202A or 8203 allow a dynamic RAM memory to be implemented using a minium of components, board space, and power, and in less design time than any other approach.

The following sections will describe each of these controllers in detail.

8202A

FUNCTIONAL DESCRIPTION

The 8202A provides total dynamic RAM control for 4K and 16K dynamic RAMs, including the Intel 2104A, 2117, and 2118. The pinout and simplified logic diagram of the 8202A are shown in Figures 10 and 11.

The 8202A is always in one of the following states:

a) IDLE
b) TEST cycle
c) REFRESH cycle
d) READ cycle
e) WRITE cycle

The 8202A is normally in the idle state. Whenever a cycle is requested, the 8202A will leave the idle state to perform the desired cycle; if no cycle requests are pending, the 8202A will return to the idle state. A refresh cycle request may originate internally or externally to

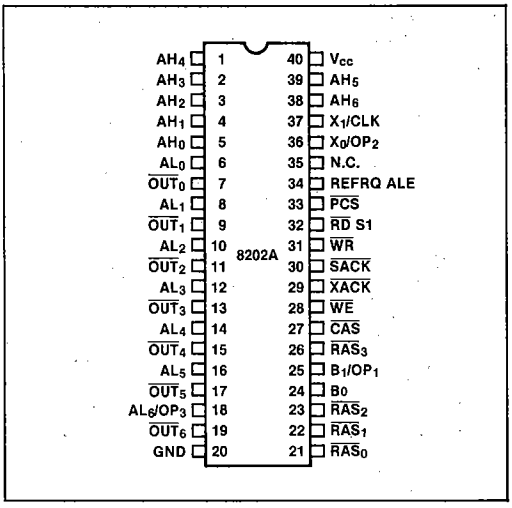


Figure 10. 8202A Pinout

the 8202A; all other requests come only from outside the 8202A.

A test cycle is requested by activating the $\overline{RD}$ and $\overline{WR}$ inputs simultaneously, independent of $\overline{PCS}$ (Protected Chip Select). The test cycle will reset the refresh address counter to zero and perform a write cycle. A test cycle should not be allowed to occur in normal system operation, as it interferes with normal RAM refresh.

A refresh cycle performs a $\overline{RAS}$-only refresh cycle of the next lower consecutive row address after the one previously refreshed. A refresh cycle may be requested by activating the REFRQ input to the 8202A; this input is latched on the next 8202A clock. If no refresh cycles are requested for a period of about 13 microseconds, the 8202A will generate one internally. By refreshing one row every 15.6 microseconds or sooner, all 128 rows will be refreshed every 2 ms. Because refresh requests are generated by the 8202A itself, memory integrity is insured, even if the rest of the system should halt operation for an extended period of time.

The arbiter logic will allow the refresh cycle to take place only if there is not another cycle in progress at the time.

A read cycle may be requested by activating the $\overline{RD}$ input, with $\overline{PCS}$ (Protected Chip Select) active. In the Advanced Read mode, a read cycle is requested if the microprocessor's S1 status line is high at the falling edge of ALE (Address Latch Enable) and $\overline{PCS}$ is active. If a dynamic RAM cycle is terminated prematurely, data loss may result. The 8202A chip select is "protected" in that once a memory cycle is started, it will go to completion, even if the 8202A becomes de-selected.

A write cycle may be requested by activating the $\overline{WR}$ input, with $\overline{PCS}$ active; this is the same for the normal and Advanced Read modes.

BLOCK DIAGRAM

Let's look at the detailed block diagram in Figure 12 to see how the 8202A satisfies the interface requirements of the dynamic RAM.

Address Multiplexing

Address multiplexing is achieved by a 3-to-1 multiplexer

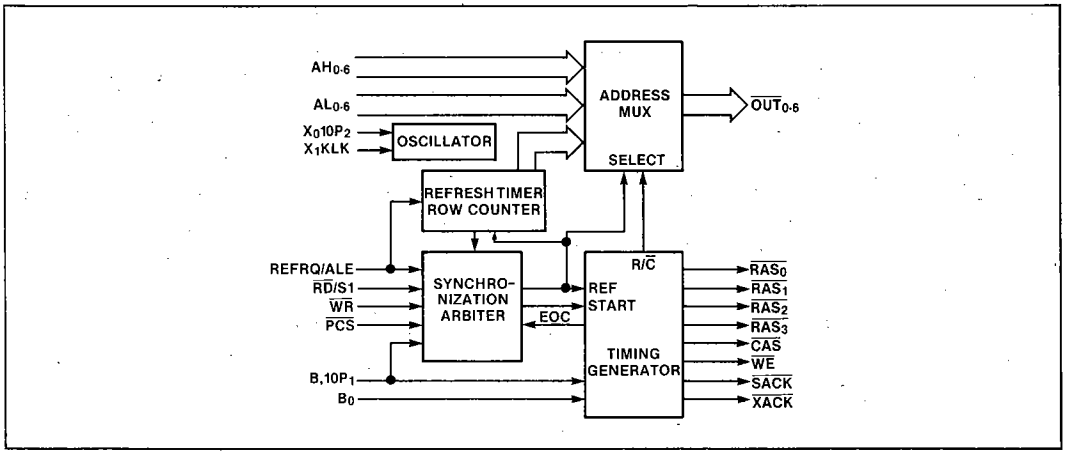


Figure 11. 8202A Simplified Block Diagram

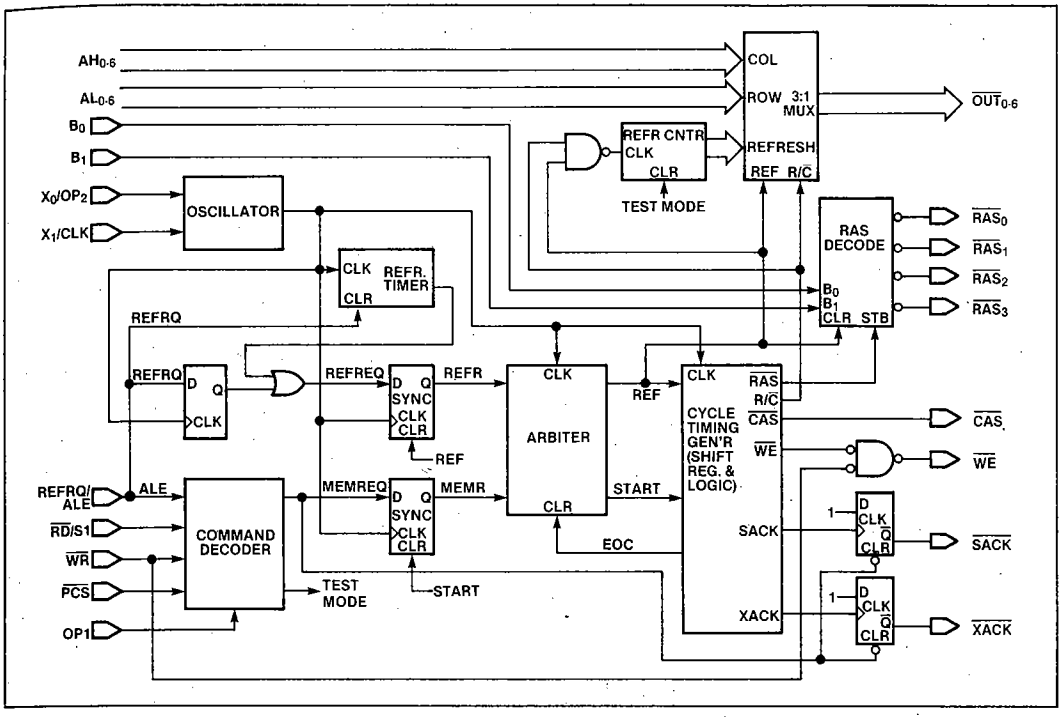


Figure 12. 8202A Detailed Block Diagram

internal to the 8202A; the three inputs are the row address ($AL_{0\text{-}6}$), column address ($AH_{0\text{-}6}$), and refresh row address (generated internally). When the 8202A is in the Idle state, the multiplexer selects the row address, so it is prepared to start a memory cycle. If a refresh cycle is requested either internally or externally, the address multiplexer will select the refresh row address long enough before $\overline{RAS}$ goes active to satisfy the RAM's t_{ASR} parameter.

To minimize propagation delays, the 8202A address outputs ($\overline{OUT}_{0\text{-}6}$) are inverted from the address inputs. This has no effect on RAM operation; inverters are not needed on the address outputs.

Doing this multiplexing internally minimizes timing skews between the address, $\overline{RAS}$, and $\overline{CAS}$, and allows higher performance than would otherwise be possible.

Refresh Counter

The next row to be refreshed is determined by the refresh counter, which is implemented as a 7-bit ripple-carry counter. During each refresh cycle, the counter is

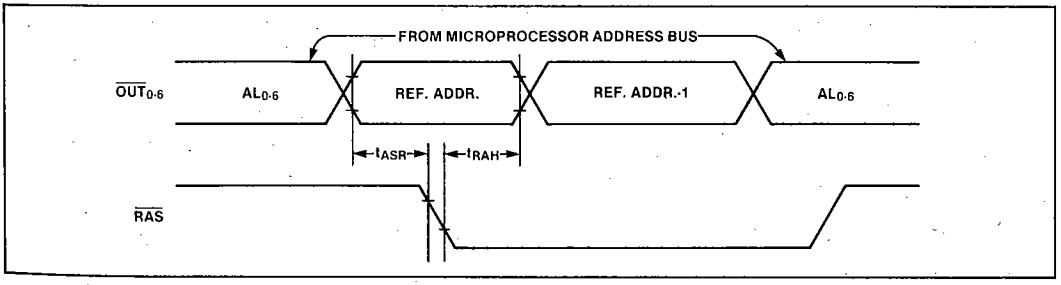


Figure 13. Detailed 8202A Refresh Cycle

incremented by one in preparation for the next refresh cycle (a refresh cycle is shown in detail in Figure 13).

When the 8202A enters TEST mode, the refresh counter is cleared. This feature is useful for automatic testing of the refresh counter function. Because the address outputs are inverted, the first refresh address after clearing the counter in test mode is $7F_H$, and the addresses *decrease* for subsequent refresh cycles.

$\overline{RAS}$ Decoding

Which bank of RAM is selected for a memory cycle is determined by the $\overline{RAS}$ decoder from the B_{0-1} inputs, which normally come from the microprocessor address bus. The 8202A Timing Generator produces an internal $\overline{RAS}$ pulse which strobes the $\overline{RAS}$ decoder, generating the appropriate external $\overline{RAS}$ pulse. The B_{0-1} inputs are *not* latched, so they must be held valid for the length of the memory cycle. During a refresh cycle, all the $\overline{RAS}$ outputs are activated, refreshing all banks at once.

Oscillator

The 8202A operates from a single reference clock with a frequency between 18.432 MHz and 25 MHz; this clock is used by the synchronization, arbitration, and timing generation logic. This clock may be generated by an on-board crystal oscillator, or by an external TTL-compatible clock source. When using the internal oscillator (available only on part number D8202A-1 or D8202A-3), a fundamental-mode crystal is attached to pins 36 and 37 (X_0 and X_1), as shown in Figure 14. The external TTL clock option is selected by pulling pin 36 (OP_2) to +12v through 1K ohm resistor, and attaching the clock input to pin 37 (CLK).

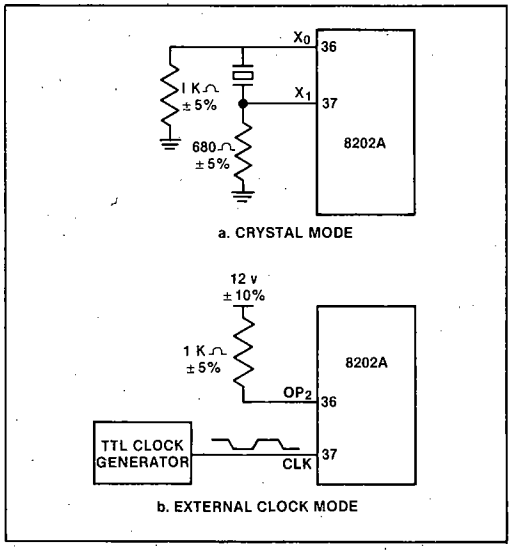


Figure 14. 8202A Clock Options

Command Decoder

The command decoder takes the commands from the bus and generates internal memory request (MEMR), and TEST signals.

The 8202A has two bus interface modes: the "normal" mode, and the "Advanced Read" mode. In the normal mode, the 8202A interfaces to the usual bus RD and WR signals.

In the Advanced Read mode, the 8202A interfaces to the Intel microprocessor bus signals ALE, S1, and $\overline{WR}$. S1 must be high on the falling edge of ALE for read cycles, and $\overline{WR}$ must be low for write cycles (write cycles are the same as for normal read mode). The 8085A S1 may be used directly by the 8202A; the 8086 and 8088 $\overline{S1}$ must be inverted. ALE and $\overline{WR}$ must be qualified by $\overline{PCS}$.

The Advanced Read mode is useful for reducing read data access time, and thus wait states. This mode is used mainly with 8085A systems.

If both $\overline{RD}$ and $\overline{WR}$ are active at once (regardless of the state of $\overline{PCS}$), the internal TEST signal is generated and the 8202A performs a test cycle as described above. One or both of $\overline{RD}$ and $\overline{WR}$ should have pull-up resistors to prevent the 8202A from inadvertantly being put into test mode, as the $\overline{RD}$ and $\overline{WR}$ signals are 3-stated by the microprocessor when RESET or HOLD are active. Since the test mode resets the refresh address counter, the refresh sequence will be interrupted, and data loss may result.

Refresh Timer and REFRQ

The 8202A contains a counter, operated from the internal clock to time the period from the last refresh cycle. When the counter times out, an internal refresh request is generated. This refresh period is proportional to the 8202A's clock period, and varies from 10.56 to 15.625 microseconds. Even at the lowest refresh rate, all the rows of the dynamic RAM will be refreshed every 2 ms.

The 8202A has an option of reducing the refresh rate by a factor of two, for use with 4K RAMS. These RAMs have only 64 rows to refresh every 2 ms, so need refresh cycles only half as often. This option is selected by pull-

ing pin 18 (AL_6/OP_3) to +12v through a 5.1K ohm resistor. This pin normally serves as the high-order row address input for the address multiplexer, but it is no longer needed for this function, as 4K RAMs have one less address input.

A refresh cycle may also be requested externally by activating the REFRQ input. This input is latched, so it only needs to be held active a maximum of 20 ns. If the 8202A is currently executing a memory cycle, it will complete that cycle, and then perform the refresh cycle. The internal and external refresh requests are ORed together before going to the arbiter.

The REFRQ input cannot be used in the Advanced Read mode, as the REFRQ pin is used for ALE in this mode.

REFRQ is most often used to implement transparent refresh, as explained in the section *Dynamic RAMS — Refresh*. This technique is not useful in iAPX 86 and iAPX 88 systems, so REFRQ is normally tied to ground.

The refresh timer is reset as soon as a refresh cycle is started (whether it was requested internally or externally). The time between refresh cycle (t_{REF}) is measured from when the first cycle is *started*, not when it was *requested*, which occurs sometime earlier. Of course, t_{REF}min does not apply if REFRQ is used — you may externally request refresh cycles as often as you wish.

Arbiter

This is the hardest section of a dynamic RAM controller to implement. If a read or write arrives at the same time as a refresh request, the arbiter must decide which one to service first. Also, if a read, write, or refresh request arrives when another cycle is already in progress, the arbiter must delay starting the new cycle until the current cycle is complete.

Both of the internal signals REFR (refresh request) and MEMR (memory cycle request) are synchronized by D-type master-slave flip-flops before reaching the arbiter. these circuits have been optimized to resolve a valid logic state in as short a time as possible. Of course, with any synchronizer, there is a probability that it will fail — not be able to settle in one logic state or the other in the allowed amount of time, resulting in a memory failure — but the 8202A has been designed to have less than one system memory failure every three years, based on operation in the worst case system timing environments.

Both synchronizers and the arbiter are operated from the 8202A's internal clock. Assuming the 8202A is initially in an idle state, one full clock period after the synchronizers sample the state of the MEMREQ and REFREQ signals, the arbiter examines the REFR and MEMR outputs of the synchronizers. If MEMR is active, the arbiter will activate START to begin the memory cycle (either read or write) on that clock. If REFR is active (regardless of the state of MEMR), the arbiter will activate START and REF to begin a refresh cycle on that clock. Once the cycle is complete, the Cycle Timing Generator will generate an end-of-cycle (EOC) signal to clear the arbiter and allow it to respond to any new or pending requests on the next clock.

Once a memory cycle is started, it cannot be stopped, regardless of the state of the $\overline{RD}$/S1, $\overline{WR}$, $\overline{ALE}$, or $\overline{PCS}$ inputs. This is necessary, as ending a dynamic RAM cycle prematurely may cause loss of data. Note, however, that the RAM $\overline{WE}$ output is directly gated by the $\overline{WR}$ input, so if $\overline{WR}$ is removed prematurely, the RAM $\overline{WE}$ pulse-width spec (t_{WP}) may be violated, causing a memory failure.

What happens if a memory request and refresh request occur simultaneously?

> If the 8202A is in the idle state, the *memory* request will be honored first.
>
> If the 8202A is *not* in the idle state (a memory or refresh cycle is in progress) then the memory cycle will lose priority and the refresh cycle will be honored first.

Remember, if the 8202A is performing a cycle, the arbiter doesn't arbitrate again until the end of that cycle. So the memory and refresh cycles are "simultaneous" if they both happen early enough to reach the arbiter before it finishes the current cycle. This arbitration arrangement gives memory cycles priority over refresh cycles, but insures that a refresh cycle will be delayed at most one RAM cycle.

Refresh Lock-Out

As a result of the 8202A operation, transparent refresh circuits like the one shown in Figure 15 should not be used. This circuit uses the RD input, with some qualifying logic, to activate REFRQ whenever the microprocessor does an opcode fetch. This circuit will work fine, as long as the 8202A never has to generate an internal refresh request, which is unlikely (if nothing else, the system RESET pulse is probably long enough that the 8202A will throw in a couple of refreshes while the microprocessor is reset). If the 8202A ever does generate its own refresh, there is a probability that the microprocessor will try to fetch an opcode while the

refresh is still in progress. If that happens, the 8202A will finish the refresh, see both the RD and REFRQ inputs active, honor the REFRQ first, and start a *second* refresh. In the meantime, the microprocessor is sitting in wait states, waiting for the 8202A to complete the opcode fetch. When the 8202A finishes the second refresh, it will see both RD and REFRQ active again, and will start a *third* refresh, etc. The system "locks up" with the microprocessor sitting in wait states *ad infinitum*, and the 8202A doing one refresh cycle after another.

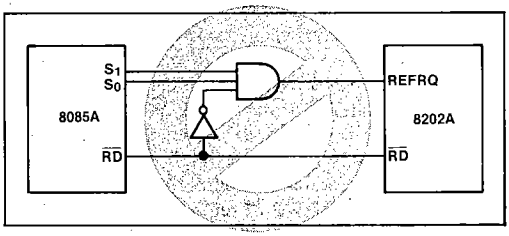


Figure 15. Improper Transparent Refresh Generation

To prevent this from happening, the transparent refresh circuit should be modified as shown in Figure 16. In this circuit, REFRQ cannot be activated until the opcode fetch is already in progress, as indicated by $\overline{SACK}$ being active (remember, $\overline{SACK}$ is never active during a refresh). If the microprocessor tries to do an opcode fetch while the 8202A is doing a refresh, REFRQ will not be active; the 8202A will finish the refresh and see only $\overline{RD}$ active, and will start the opcode fetch; only *then* will REFRQ be activated.

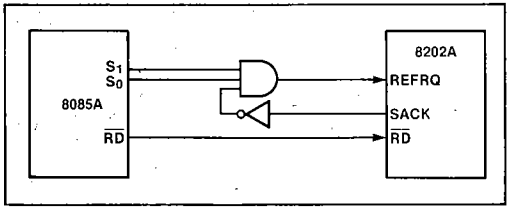


Figure 16. Generating Transparent Refresh For 8085A Systems

Cycle Timing Generator

The Cycle Timing Generator consists of a travelling-ones shift register and combinational logic required to generate all the RAM control signals and $\overline{SACK}$ and $\overline{XACK}$. All timings are generated from the 8202A's internal clock; no external delay lines are ever needed. The timing of these signals relative to CLK is illustrated in Figure 17.

When the cycle is complete, the Cycle Timing Generator sends an end-of-cycle (EOC) pulse to the arbiter to enable it to respond to new or pending cycle requests.

Minimum and maximum values for the 8202A parameters t_{CR} (Command to $\overline{RAS}$ active delay) and t_{CC} (Command to $\overline{CAS}$ active delay) differ by one 8202A clock period. This is because the commands ($\overline{RD}$, $\overline{WR}$, ALE) must be synchronized to the 8202A's clock; this introduces a ± one clock period (t_p) uncertainty due to the fact that the command may or may not be sampled on the first clock after it goes active, depending on the set-up time. If $\overline{RD}$ or ALE and $\overline{WR}$ are synchronous to the 8202A's clock, and the set-up time (t_{SC}) is met, the smaller number of clock periods will apply.

All 8202A output timings are specified for the capacitive loading in the data sheet. Typical output characteristics are shown in the data sheet for capacitive loads ranging from 0 to 660 pF, these can be used to calculate the effect of different loads than those specified in the data sheet on output timings. All address, $\overline{RAS}$, $\overline{CAS}$, and $\overline{WE}$ drivers are identical, so these characteristic curves apply to all outputs.

$\overline{SACK}$ AND $\overline{XACK}$

Because refresh cycles are performed asynchronously to the microprocessor's operation (except during transparent refresh), the microprocessor cannot know when it activates $\overline{RD}$ or $\overline{WR}$ if a refresh cycle is in progress, and therefore, it can't know how long it will take to complete the memory cycle.

This added consideration requires an acknowledge or "handshake" signal from the 8202A to tell the microprocessor when it may complete the memory cycle. This acknowledge would be used to generate the microprocessor's READY input — the microprocessor will sit in wait states until the 8202A acknowledges the memory cycle. Two signals are generated for this purpose by the 8202A; they are called *system acknowledge* ($\overline{SACK}$) and *transfer acknowledge* ($\overline{XACK}$). They serve the same purpose but differ in timing.

$\overline{XACK}$ is a Multibus-compatible signal, and is not activated until the read or write cycle has been completed by the RAMs. In a microprocessor system, however, there is a considerable delay from when the 8202A acknowledges the memory cycle until the microprocessor actually terminates the cycle. This delay is due to the time required to combine this acknowledge with other sources of READY in the system, synchronize READY to the microprocessor's clock, sample the state of READY, and respond to an active READY signal. As a result, more wait states than necessary may actual-

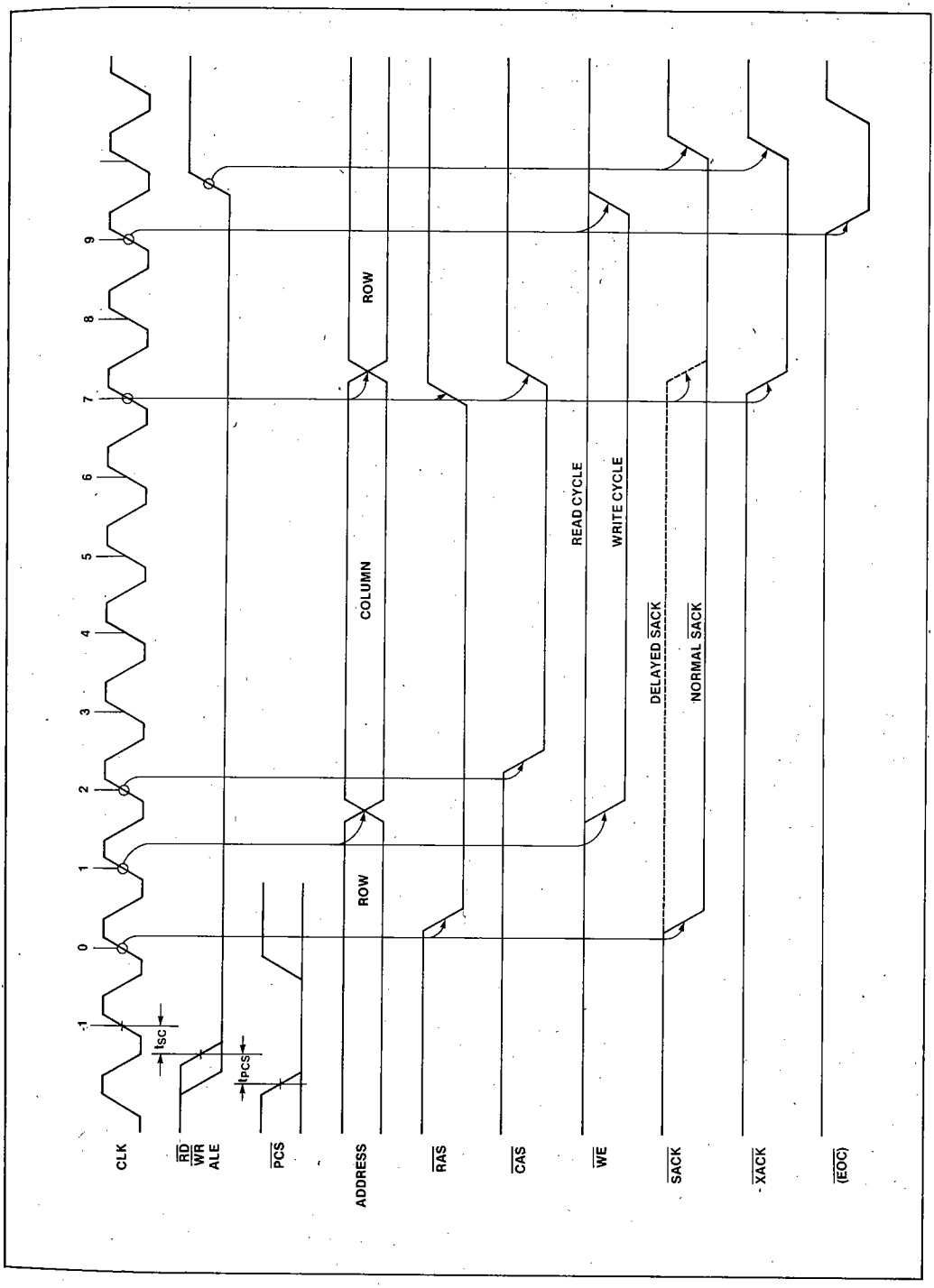


Figure 17. 8202A Timing Relative To CLK

ly be generated by using $\overline{XACK}$. $\overline{SACK}$ is activated earlier in the cycle to improve performance of microprocessors by compensating for the delays in the microprocessor responding to $\overline{XACK}$, and thus eliminating unneeded wait states which might be generated as a result of $\overline{XACK}$ timing. The system designer may use one or the other acknowledge signal, or use both in different parts of the system, at his option.

$\overline{SACK}$ and $\overline{XACK}$ are activated by the Cycle Timing Generator, but they can be de-activated only by the microprocessor removing its $\overline{RD}$ or $\overline{WR}$ request, or by activating ALE when in the advanced read mode. As the $\overline{SACK}$ and $\overline{XACK}$ signals are used to generate READY for the microprocessor, this is necessary to give the microprocessor as much time as it needs to respond to its READY input.

Delayed SACK Mode

$\overline{SACK}$ may be activated at one of two different times in the memory cycle; the earlier case is called "normal $\overline{SACK}$" and the later is called "delayed $\overline{SACK}$" (Figure 18). Delayed $\overline{SACK}$ occurs if the memory request was received by the 8202A while it was doing a refresh cycle. In this case, the memory cycle will be delayed some length of time while the refresh cycle completes; $\overline{SACK}$ is delayed to ensure the microprocessor will generate enough wait states. This is a concern mostly for read cycles.

Because of the way the delayed $\overline{SACK}$ mode is implemented in the 8202A, if the $\overline{RD}$ or $\overline{WR}$ input is activated while a refresh cycle is in progress, regardless of whether or not the 8202A is chip-selected, the internal delayed $\overline{SACK}$ mode flip-flop will be set. The next 8202A memory cycle will have $\overline{SACK}$ delayed, even if that cycle was not actually delayed due to a refresh cycle in progress. The delayed SACK flip-flop will be reset at the end of that cycle, and the 8202A will return to normal $\overline{SACK}$ operation. The same thing happens in Advanced Read mode if S1 is high at the falling edge of ALE during a refresh cycle, once again regardless of the state of $\overline{PCS}$.

8203

The 8203 is an extension of the 8202A architecture which allows the use of 64K dynamic RAMs. It is pinout compatible with the 8202A and shares identical A.C. and D.C. parameters with that part. The description of the 8202A applies to this part also, with the modifications below.

ENHANCEMENTS

1. Supports 16K *or 64K* dynamic RAMs. 4K RAM mode, selected by pulling AL_6/OP_3 (pin 18) to +12v, is not supported.

2. Allows a single board design to use either 16K or 64K RAMs, without changing the controller, and only making between two and four jumper changes to reconfigure the board.

3. May operate from external TTL clock without the +12v pull-up which the 8202A requires (a +5v or +12v pull-up may be used).

The pinout of the 8203 is shown in Figure 19. This pinout is identical to the 8202A, with the exception of the five highlighted pins. The function of these is described below. The simplified block diagram is similar to the 8202A's, in Figure 11.

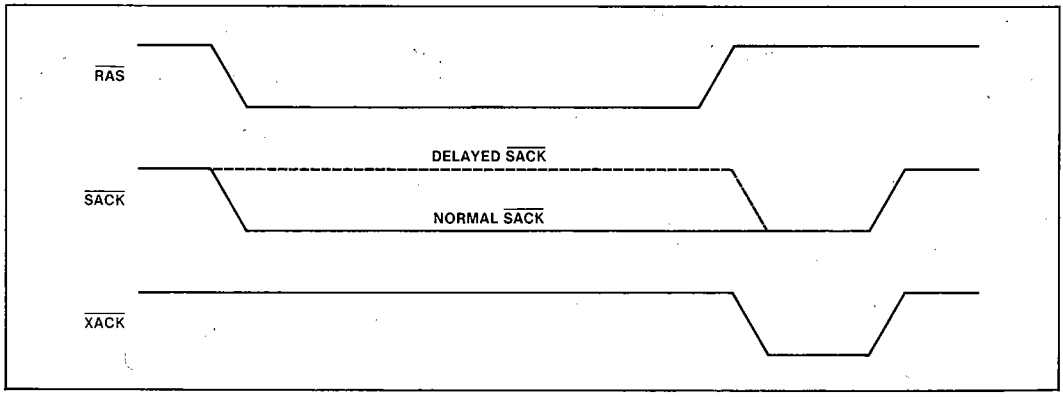


Figure 18. Delayed SACK Mode

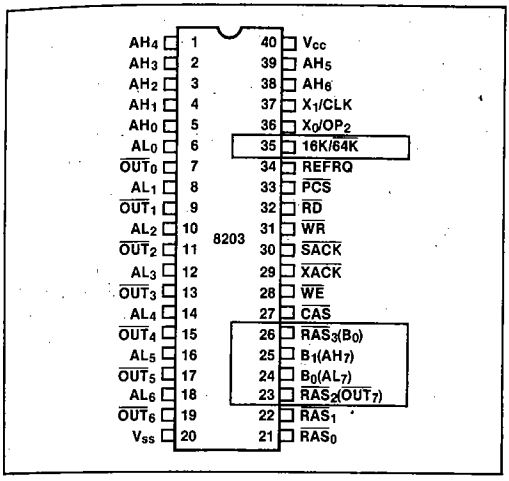


Fig. 19 8203 Pinout

16K Mode and 64K Mode

The goal of the 8203 is to provide a pin- and timing-compatible upgrade of the 8202A for use with 64K RAMs. The difficulty in doing this is that 64K RAMs require an additional address input compared to 16K RAMs, and thus the 8203 needs three more pins (one more RAM address output, and two more inputs to its internal address multiplexer). Since all but one of the 8202A's pins are already used, this is clearly a challenge — some functionality must be sacrificed to gain 64K RAM support. The 8203 reduces the maximum number of banks supported from four to two for 64K RAMs.

Pin 35 (16K/$\overline{64K}$) is used to tell the 8203 whether it is being used to control 16K RAMs or 64K RAMs. When tied to V_{cc} or left unconnected, the 8203 operates in the 16K RAM mode; in this mode all the remaining pins function identically to the 8202A. When tied to ground, it operates in the 64K RAM mode, and pins 23 through 26 change function to enable the 8203 to support 64K RAMs. Pin 35 (16K/$\overline{64K}$) contains an internal pull-up —when unconnected, this input is high, and the 8203 operates identically to the 8202A. This maintains pinout compatibility with the 8202A, in which pin 35 is a no-connect, so the 8203 may be used in 8202A sockets with no board modifications.

When the 8203 is in the 64K RAM mode, four pins change function, as shown in Table 2. The pins change function in this particular way to allow laying out a board to use either 16K or 64K RAMs with a minimum of jumpers, as shown in Figure 20. This figure shows the 8203 with two banks of RAM. Banks 0 and 1 may be either 16K RAMs or 64K RAMs; banks 2 and 3 may only be 16K RAMs, as the 8203 supports two banks of 64K RAM. For clarity, only those connections which are important in illustrating the 8203 jumper options are shown.

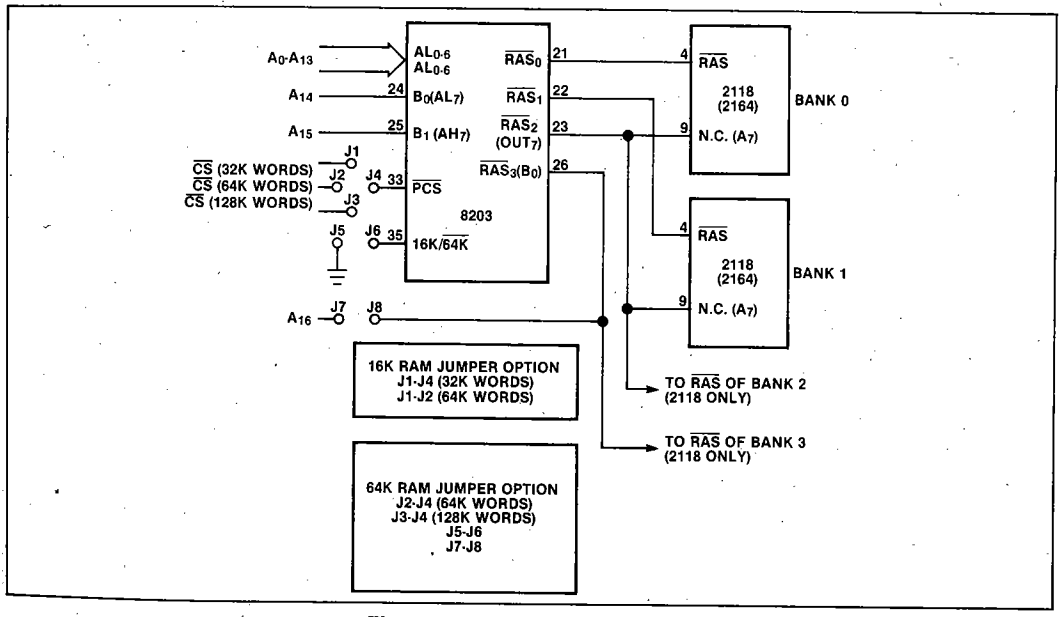


Figure 20. 8203 Jumper Options

Table 2. 16K/$\overline{64K}$ Mode Selection

Pin #	16K Function	64K Function
23	$\overline{RAS}_2$	Address Output (OUT_7)
24	Bank Select (B_0)	Address Input (AL_7)
25	Bank Select (B_1)	Address Input (AH_7)
26	$\overline{RAS}_3$	Bank Select (B_0)

Jumpers J1-J4 may be used to chip select the 8203 over various address ranges. For example, if two banks of 16K RAMs are replaced with two banks of 64K RAMs, the address space controlled by the 8203 increases from 32K words to 128K words. If four banks of 16K RAMs are replaced with one bank of 64K RAMs, no chip select jumpers are needed.

In the 64K RAM mode, pins 24 and 25 ($B_0(AL_7)$ and $B_1(AH_7)$) change function from bank select inputs to address inputs for the 64K RAM. Since the bank select inputs normally come from the address bus anyway, no jumper changes are required here. The bank select function moves to pin 26 ($\overline{RAS}_{3(B_0)}$); since only two bank of 64K RAM is supported, only one bank select input is needed in this mode, not two. Jumpers J6 and J7 are shorted in the 64K RAM mode to connect pin 26 (B_0) to the address bus. In the 16K RAM mode, these jumpers must be disconnected, as pin 26 junctions as the $\overline{RAS}_3$ output; in the 64K RAM mode, this bank is not populated, so $\overline{RAS}_3$ is not needed.

Pin 23 serves two functions: in the 16K RAM mode it is the $\overline{RAS}$ output for bank 2 ($\overline{RAS}_2$), in the 64K RAM mode is the high order RAM address output ($\overline{OUT}_7$), which goes to pin 9 of the 64K RAMs. This requires no jumpers as when using 16K RAMs, pin 9 is a no-connect, and when using 64K RAMs, bank 2 is depopulated, so $\overline{RAS}_2$ is not used.

This arrangement allows converting a board from 16K RAMs to 64K RAMs with no change to the controller and changing a maximum of three jumpers.

+5v External Clock Option

Just as with the 8202A, the user has the option of an external TTL clock instead of the internal crystal oscillator as the timing reference for the 8203; unlike the 8202A, he does not need to tie pin 36 ($X_0/0P_2$) to +12v to select this option—this pin may be tied to *either* +5v *or* +12v. If pin 36 is tied to +12v, a 1K ohm (± 5%) series resistor must be used, just as for the 8202A. If pin 36 is tied to +5v, it must be tied *directly* to pin 40 (V_{cc}) with no series resistor. This is because pin 36 must be within one Schottky diode voltage drop (roughly 0.5v) of pin 40 to select the external TTL clock option; a series resistor may cause too great a voltage drop for the external clock option to be selected. For the same reason, the trace from pin 36 to 40 should be kept as short as practical.

Test Cycle

An 8203 test cycle is requested by activating the $\overline{RD}$, $\overline{WR}$, and $\overline{PCS}$ inputs simultaneously. By comparison, an 8202A test cycle requires activating only the $\overline{RD}$ and $\overline{WR}$ inputs simultaneously, independent of $\overline{PCS}$. Like the 8202A, and 8203 test cycle resets the address counter to zero and performs a write cycle.

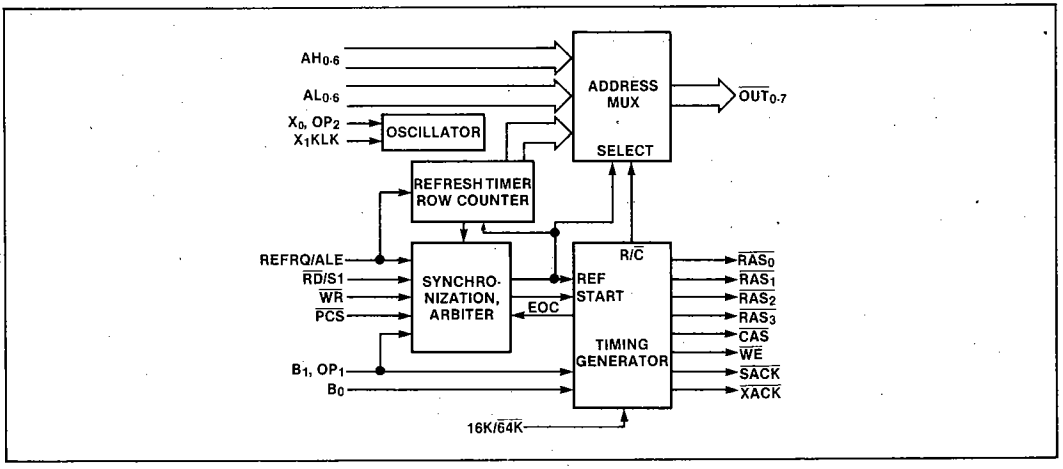


Figure 21. 8203 Simplified Block Diagram

BLOCK DIAGRAM

A simplified block diagram of the 8203 is shown in Figure 21. It is identical to the 8202A except for the following differences:

1. The 3:1 address multiplexer is 8 bits wide, instead of 7 bits wide, to support the addressing requirements of the 64K RAM.
2. The refresh address counter is 8 bits. This allows it to support RAMs which use either the 128-row or 256-row refresh schemes. Regardless of which type of RAM is used, the refresh counter cycles through 256 rows every 4 ms. RAMs which use 128-row re-fresh treat the eighth address bit as a "don't care" during refresh, so they see the equivalent of 128-row refresh every 2 ms. In either case the rate of internally-generated refresh cycles is the same—at least one every 15.6 microseconds.

INTEL iAPX-86 AND iAPX-88

Device Descriptions

The iAPX-86 and iAPX-88 are advanced 16-bit microprocessor families, based on the 8086 and 8088 microprocessors, respectively. While both have a similar architecture and are software compatible, the 8086 transfers data over a 16-bit bus, while the 8088 uses an 8-bit data bus (but has a 16-bit internal bus).

Min and Max Modes

In order to support the widest possible range of applications, the 8086 and 8088 can operate in one of two modes, called minimum and maximum modes. This allows the user to define certain processor pins to "tailor" the 8086 or 8088 to the intended system. These modes are selected by strapping the MN/MX (minimum/maximum) input pin to V_{CC} or ground.

In the minimum mode, the microprocessor supports small, single-processor systems using a minimum of components. In this mode, the 8086 or 8088 itself generates all the required bus control signals (Figure 22).

In the maximum mode, the microprocessor supports larger, higher performance, or multiprocessing systems. In this mode, the 8086 or 8088 generates status outputs which are decoded by the Intel 8288 Bus Controller to provide an extensive set of bus control signals, and Multibus compatibility (Figure 23). This allows higher performance RAM operation because the memory read and write commands are generated more quickly than is possible in the minimum mode. The maximum mode is the one most often used in iAPX-86 and iAPX-88 systems.

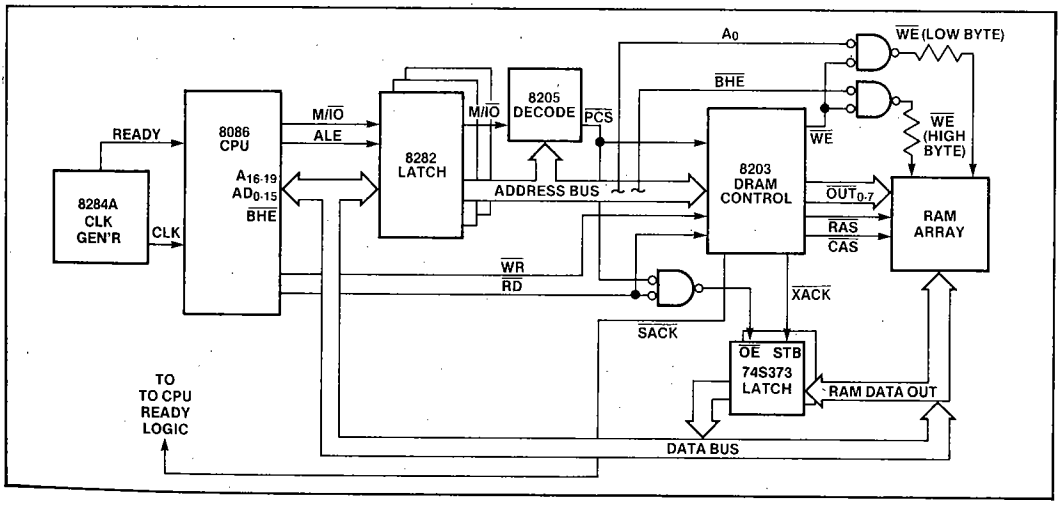


Figure 22. 8086 Minimum Mode

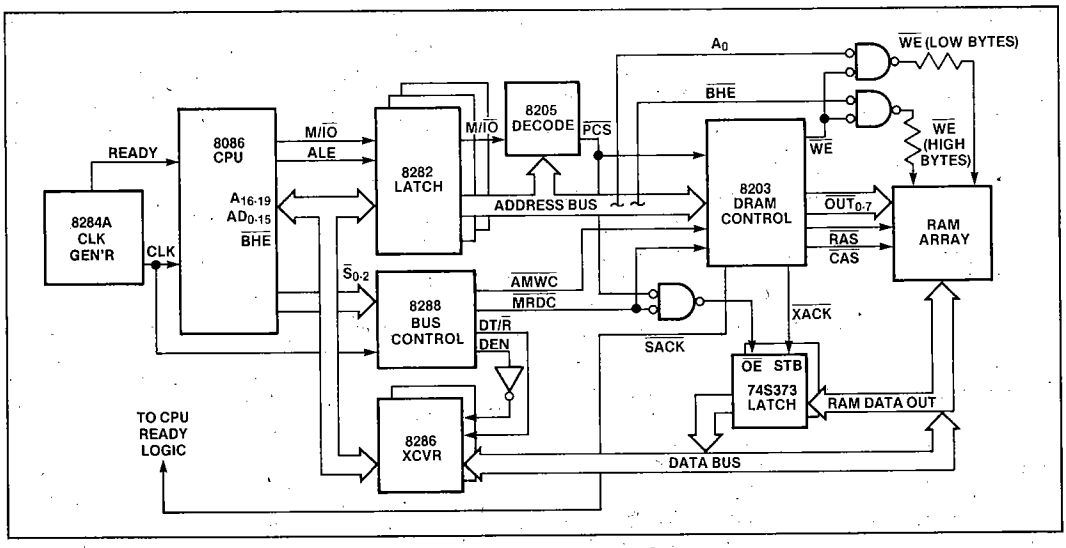


Figure 23. 8086 Maximum Mode

Alternate Configuration

The Alternate Configuration is not an operating mode of the 8086 or 8088 *per se*, but uses TTL logic along with the status outputs of the microprocesor to generate the RAM read and/or write control signals (Figure 24). The alternate configuration may be used with the microprocessor in either minimum or maximum mode. This configuration is advantageous because it activates the memory read and write signals even earlier than the maximum mode, leading to higher performance. It is possible to generate either the RAM read or write signal using this configuration, and generate the other RAM control signal using the min or max mode in the normal configuration.

Each of the three system configurations may be used with buffers on the address, data, or control bus for increased electrical drive capability.

Performance vs. Wait States

Before starting a discussion of timing analyses, it's worthwhile to look at the effect of wait states on the iAPX-86 and iAPX-88.

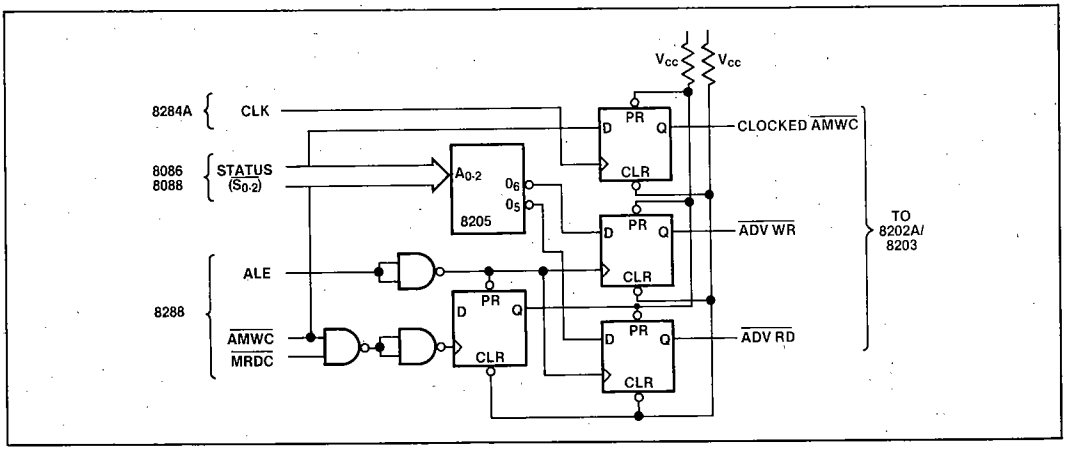


Figure 24. Alternate Configuration Logic

For most microprocessors, the effect of, say, one wait state on execution times is straightforward. If a bus cycle normally is three clocks long, adding a wait state to every bus cycle will make all bus cycles four clocks, decreasing performance by 33%. This is multiplied by the percentage of time that the microprocesor is doing bus cycles (some instructions take a long time to execute, so the microprocessor skips a few bus cycles).

The effect of wait states on the iAPX-86 and iAPX-88 is not so straightforward, however.

The 8086 and 8088 microprocessors consist of two processing units: the execution unit (EU) executes instructions, and the bus interface unit (BIU) fetches instructions, reads operands, and writes results. During periods when the EU is busy executing instructions, the BIU "looks ahead" and fetches more instructions from the next consecutive addresses in memory; these are stored in an internal queue. This queue is four bytes long for the 8088 and six bytes long for the 8086; under most conditions, the BIU can supply the next instructions without having to perform a memory cycle. Only when the program doesn't proceed serially (e. g. a Jump or Call instruction) does the EU have to wait for the next instruction to be fetched from memory. Otherwise, the instruction fetch time "disappears" as it is proceeding in parallel with execution of previously fetched instructions. The EU then has to wait for the BIU only when it needs to read operands from memory or write results to memory. As a result, the 8086 and 8088 are less sensitive to wait states than other microprocessors which don't use an instruction queue. The effect of wait states on 8086 execution time compared to the Motorola 68000 and Zilog Z8000 for a typical mix of software is summarized in Table 3.[1]

Table 3. Effects of Wait States on Execution Time

Processor	Execution Time Increase Over 0 Wait State Execution Time		
	1 Wait State	2 Wait States	3 Wait States
iAPX 86/10 (measured)	8.3%	16.3%	26.3%
Z8000 (computed)	19.1%	38.2%	57.3%
68000 (computed)	15.9%	31.9%	47.8%

The BIU can fetch instructions faster than the EU can execute them, so wait states only affect performance to the extent that they make the EU wait for the transfer of operands and results. How much this affects program execution time is a function of the software; programs that contain many complex instructions like multiplies and divides and register operations are slowed down less than programs that contain primarily simple instructions. The effect of wait states on the 8086 and 8088 is always less than on other microprocessors which don't use an instruction queue.

[1] From *16-Bit Microprocessor Benchmark Report: iAPX-86, Z8000, and 68000,* publ. by Intel Corp. 1980

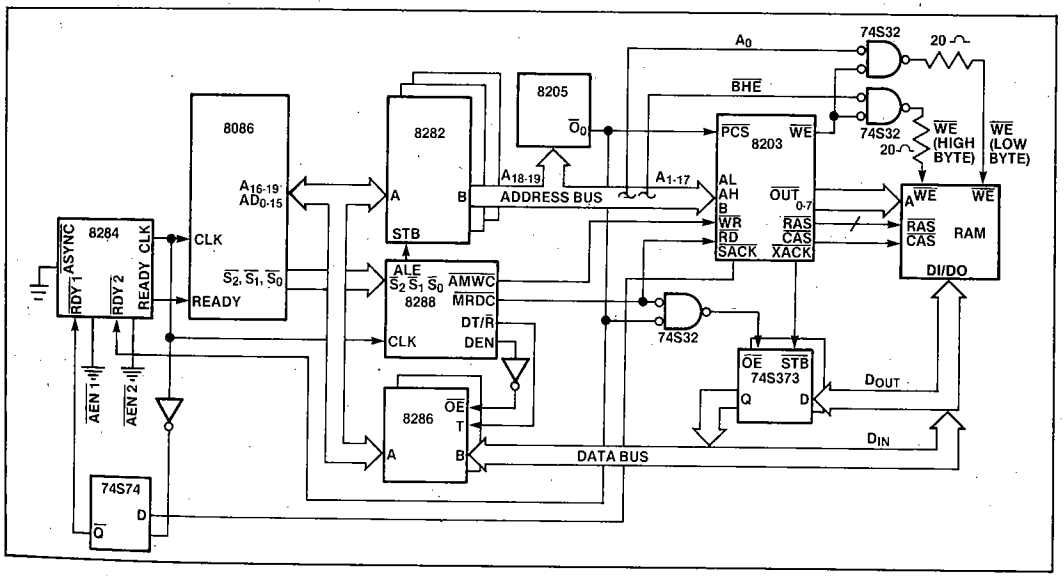


Figure 25. 8086 Max Mode System

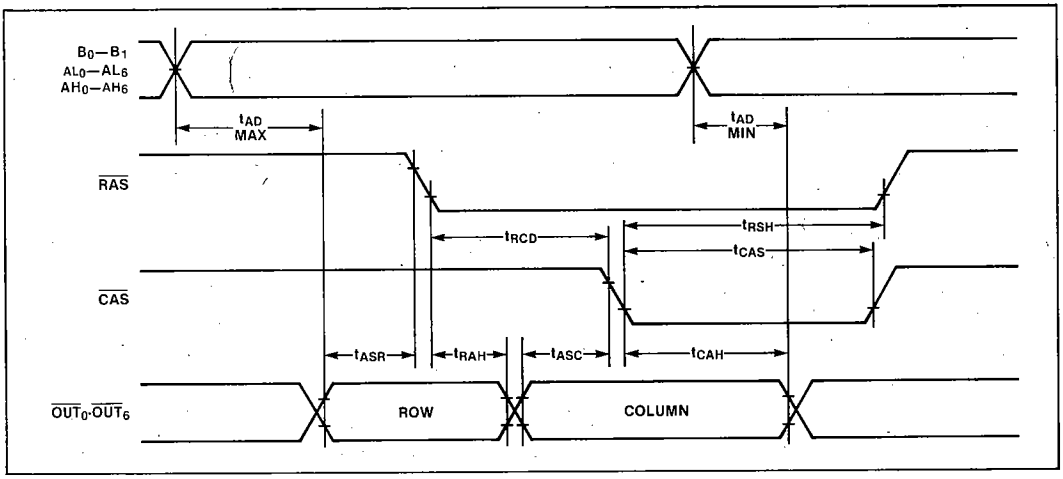


Figure 26. Memory Compatibility Timing

Timing Analysis

This section will look at two specific system configurations to show how the 8203 timing requirements are satisfied by the 8086. Methods of determining the worst case number of wait states for the various configurations are also given.

The timings of the 8202A and 8203 are identical; only the 8203 is referred to for the remainder of this note, but all comments apply equally to the 8202A. A11 timings are worst case over the range of T_A = 0 - 70°C and V_{cc} = + 5v ± 10% for the test conditions given in the devices' data sheets.

Example 1. 8086 Max Mode System (5 MHz)

This example (Figure 25) is representative of a typical medium-size microprocesor system. Example 1 requires one wait state (worst case) for memory cycles. Example 2 also uses an 8086 in Max mode at 5 MHz, but uses external logic to reduce the number of wait states to zero for both read and write cycles.

DYNAMIC RAM INTERFACE

First, look at the timing requirements of the dynamic RAM to ensure they are satisfied by the 8203. Memory compatibility timings are shown in the 8203 data sheet (Figure 26). Seven 8203 timings are given, not counting t_{AD}, which will be discussed in the next section. These timings are summarized in Table 4.

Table 4. Memory Compatibility Timings
(all parameters are minimums)

Symbol	Parameter	Value
t_{ASC}	Column Address Set-Up Time	t_p-30
t_{ASR}	Row Address Set-Up Time	t_p-30
t_{CAH}	Column Address Hold Time	$5t_p$-30
t_{CAS}	$\overline{CAS}$ Pulse Width	$5t_p$-10
t_{RAH}	Row Address Hold Time	t_p-10
t_{RCD}[1]	$\overline{RAS}$ to $\overline{CAS}$ Delay Time	$2t_p$-40
t_{RSH}	$\overline{RAS}$ Hold Time from $\overline{CAS}$	$5t_p$-30

[1] $t_{RCD}min = t_{RAH}min + t_{ASC}min = 2_p - 40$
This parameter is the minimum RAS active to CAS active delay.

These timings are all a function of the 8203's clock period (t_p); they may be adjusted to be compatible with slower dynamic RAMs by slowing the 8203's clock (increasing t_p). The frequency of the 8203's clock may be varied from 18.432 MHz to 25 MHz; for best performance, the 8203 should be operated at the highest possible frequency compatible with the chosen dynamic RAM. In most cases, t_{RAH} or t_{CAS} will be the frequency limiting parameter, but the 8203 can operate at its maximum frequency with most dynamic RAMs available.

t_{ASR} applies only to refresh cycles. When the 8203 is in the Idle state (not performing any memory or refresh cycles) the address multiplexer allows the AL_{0-7} inputs (the RAM row address) to propagate through to the 8203 OUT_{0-7} pins, which are connected to the RAM address pins. So in read or write cycles, the row address will propagate directly from the address bus to the

RAM; the row address set-up time in this case is determined by the microprocessor's timing (see the next section). At the beginning of a refresh cycle, the 8203 has to switch its internal multiplexer to direct the refresh row address to the RAMs before activating RAS; the t_{ASR} parameter in Table 4 refers to this case only.

Assume the Intel 2164A-20 RAM (200 ns access time) is used. Equations 1(a)-(h) show that this RAM is compatible at the 8203's maximum operating frequency of 25 MHz (t_p = 1/(25 MHz) = 40 ns). This frequency will be used for now; once the rest of the system timings are calculated, the minimum 8203 frequency which will provide the same system performance can also be determined.

(a) $t_{ASC} = t_p - 30 = 10$ (Equation 1.)
(b) $t_{ASR} = t_p - 30 = 10$
(c) $t_{CAH} = 5t_p - 30 = 170$
(d) $t_{CAS} = 5t_p - 10 = 190$
(e) $t_{RAH} = t_p - 10 = 30$
(f) t_{RCD}[1] $= 2t_p - 40 = 40$
(g) $t_{RP} = 4t_p - 30 = 130$
(h) $t_{RSH} = 5t_p - 30 = 170$

[1] May be calculated as
$t_{RCD}min = t_{RAH}min + t_{ASC}min = 2t_p - 40$

ADDRESS SET-UP AND HOLD TIME MARGINS

The microprocessor must put the memory address on the address bus early enough in the memory cycle for it to pass through the 8203 and meet the row address set-up time to RAS (t_{ASR}) requirement of the dynamic RAM (Figure 27). Since the address propagates directly through the 8203, this set-up time is a function of how long the microprocessor holds the address on the bus before activating the RD or WR command, the address delay through the 8203 (t_{AD}max), and how long the 8203 waits before activating RAS (t_{CR}min). This is shown in Figure 28, and calculated in Equation 2. This and all following equations show timing margins; a positive result indicates extra margin, a zero result says the parameter is just met, and a negative result indicates it is not met for worst-case conditions.

Row Address Set-Up Time Margin (Equation 2.)
= CPU Address to $\overline{RD}$ Delay + $\overline{RAS}$ Active Delay - Address Delays
= TCLCL(5MHz) + TCLML min (8288) + t_{CR}min(8203) - [Greater of TCLAVmax(8086) + TIVOVmax (8282) or TCLLHmax(8288) + TSHOVmax(8282)] - t_{AD}max(8203) - t_{ASR}(2164A—20)
= 200 + 10 + [40 + 30] - [Greater of (110 + 30) or (15 + 45)] - 40 - 0
= 100

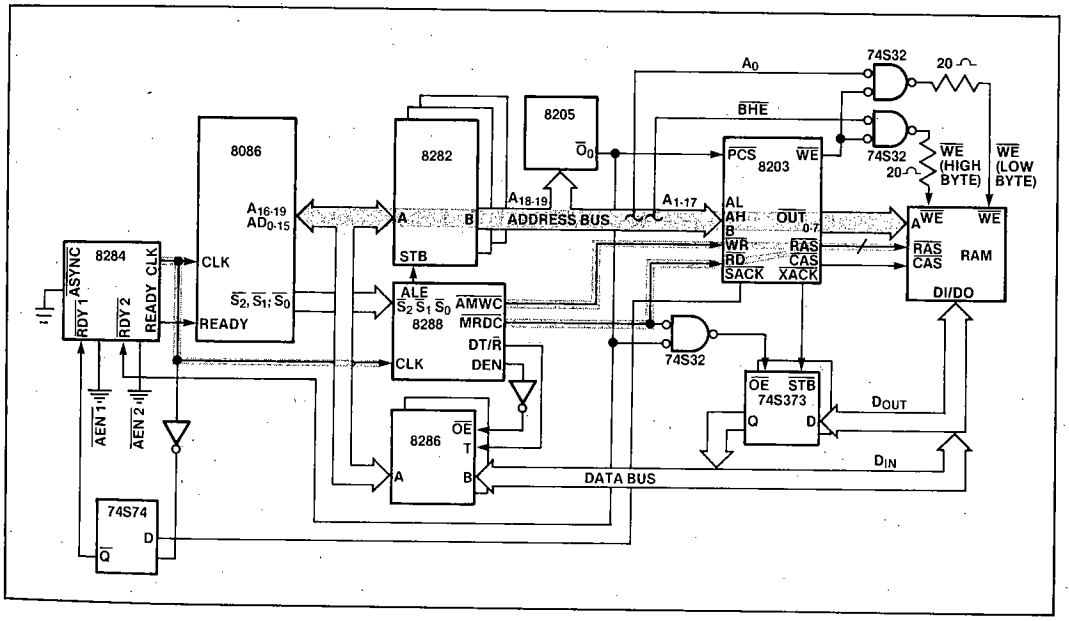


Figure 27. Address Set-Up and Hold Time Margins

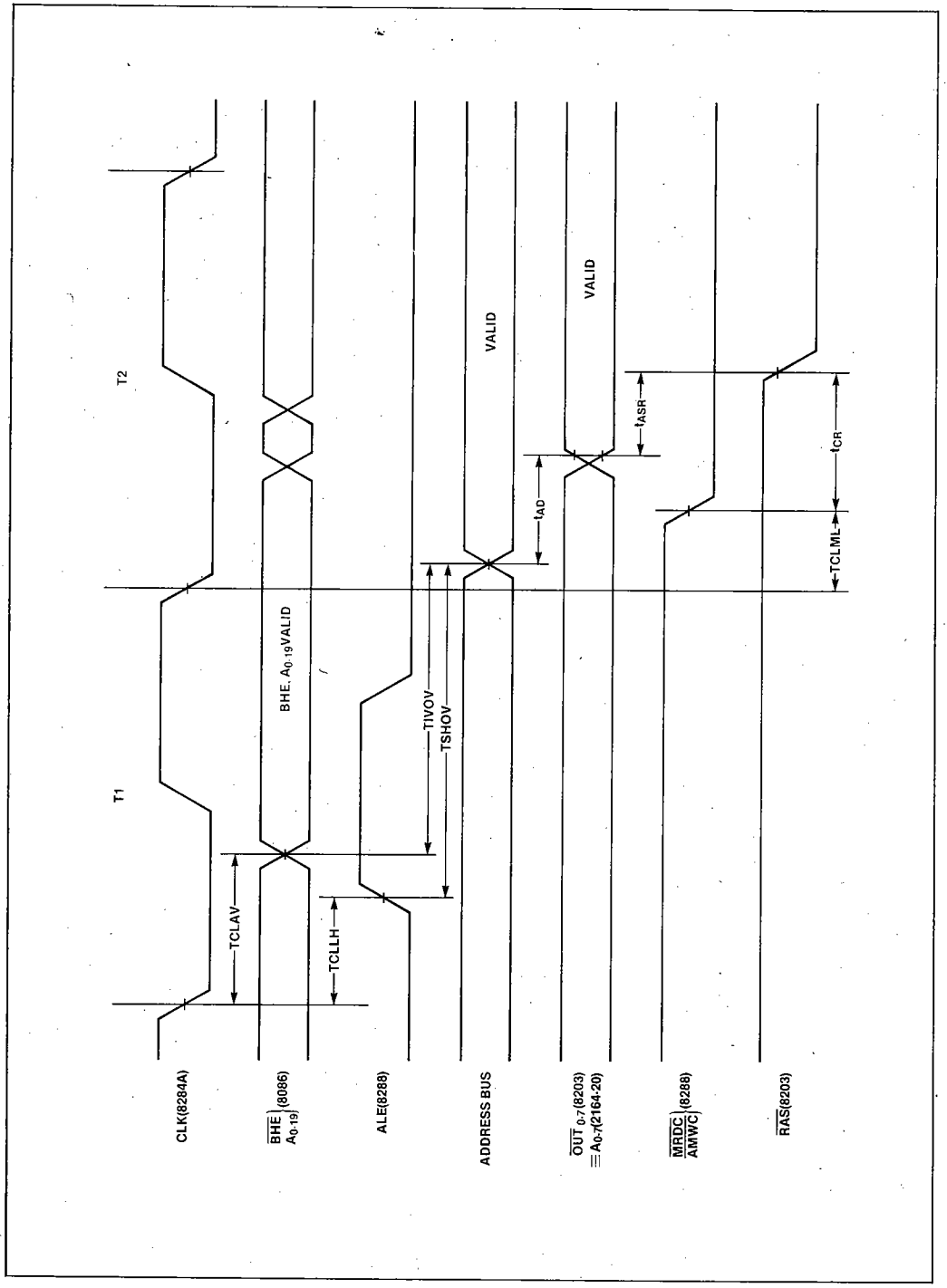


Figure 28. Address Set-up Time Margin

210398-001

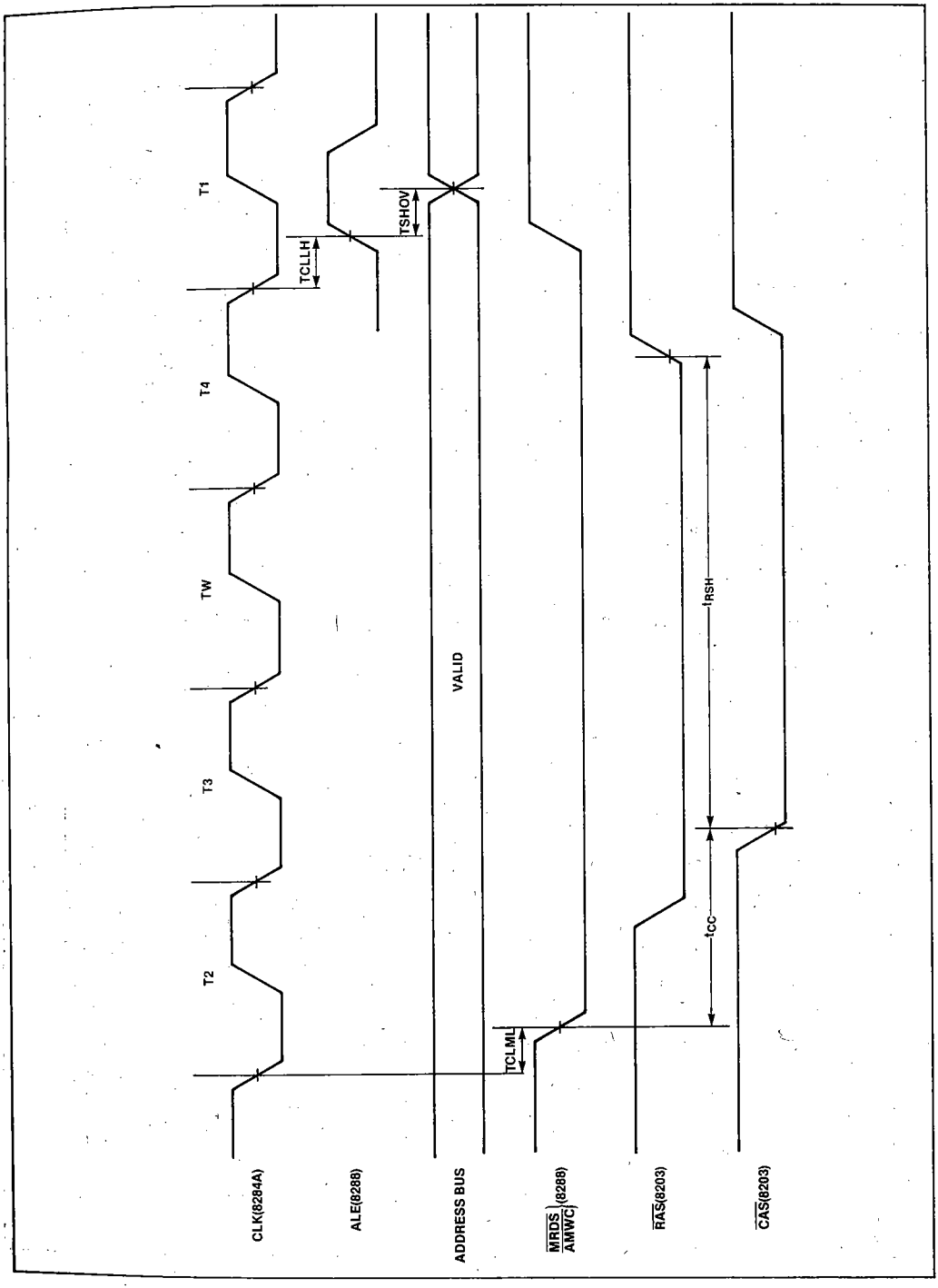


Figure 29. Address Hold Time Margin

Similarly, the microprocessor must maintain the memory address long enough to satisfy the column address hold time (t_{CAH}) of the RAM; the 8203 T_{AD}min parameter should be used for this calculation.

More importantly, the 8203 bank select (B_{0-1}) inputs are also not latched; these are used directly to decode which $\overline{RAS}$ output is activated during read or write cycles, so these inputs must be held valid until $\overline{RAS}$ goes inactive. Since B_{0-1} are usually taken directly from the address bus, this determines the address hold time required of the system (Figure 29). These are easily satisfied by the 8086 as shown by Equation 3. N represents the number of wait states. This equation can be tried with various values for N (starting with 0 and increasing) until the equation is satisfied, or it can be set equal to zero (meaning no excess margin remains) and solved for N directly; the fractional value for N that results must be rounded *up* to get the worst-case number of wait states to satisfy this particular parameter. No wait states are required to meet address hold times.

Address Hold Time Margin (N = 0) (Equation 3.)

= CPU Address Hold Time, from $\overline{RD}$ Active - $\overline{RAS}$ Inactive Delays

= (3 + N)TCLCL(5MHz) + TCLLHmin(8288)[1] + TSHOVmin(8282) - TCLMLmax(8288) - t_{CC}mas(8203) - t_{RSH}max(8203)[2]

= 3(200) + 2 + 10 - 35 - [4(40) + 85] - [5(40) + 30]

= 102

READ DATA ACCESS TIME MARGIN

Read data access times determine how many wait states are required for read cycles. Remember that dynamic RAMs have two access time parameters, $\overline{RAS}$ access time (t_{RAC}) and $\overline{CAS}$ access time (t_{CAC}). Either one may be the limiting factor in determining RAM access time, as explained in the section *Dynamic RAM - Access Times*, above. Here t_{CAC} is the limiting factor, as

$$t_{CC}max + t_{CAC}max \geq t_{CR}max + t_{RAC}max.$$

This timing is shown in Figures 30 and 31, and is calculated in Equation 4. In this system, one wait state is required to satisfy the read data access time requirements of the system; the margin is -50 ns, which is too large a difference to be made up by using a faster RAM.

[1] Not specified — use 2 ns

[2] Not specified in 8203 data sheet; t_{RSH}max(8203) = $5t_p$ + 30

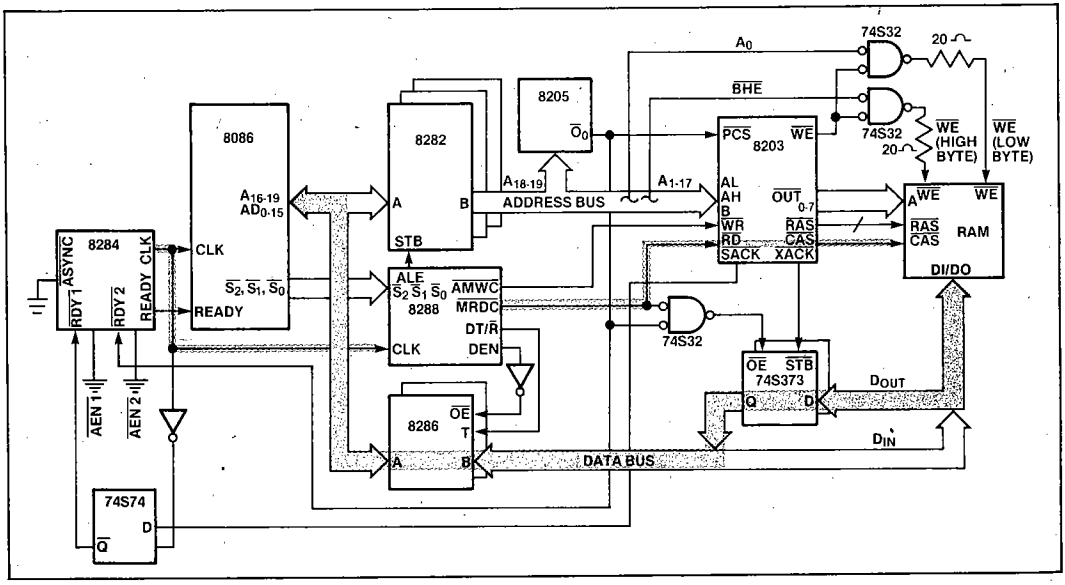


Figure 30. Read Data Access Time Margin

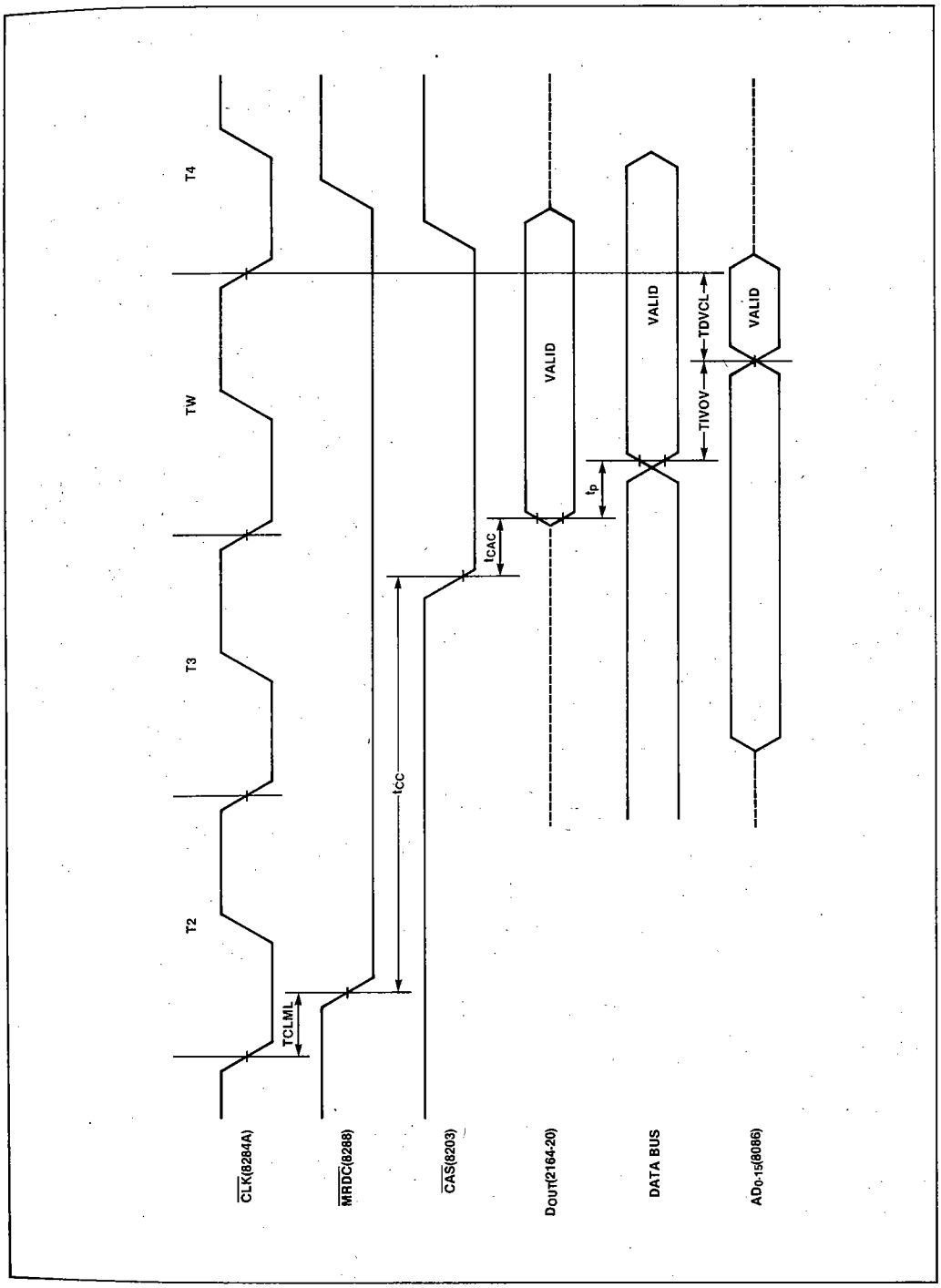


Figure 31. Read Data Access Time Margin

Read Data Access (Equation 4.)
Time Margin (N = 0)

= CPU $\overline{RD}$ Active to Data Valid Delay - $\overline{CAS}$ Active Delay - Data Delays

= (2+N)TCLCL(5MHz) - TCLMLmax(8288) t_{CC}max(8203) - t_{CAC}max(2164A—20) - $t_P$max(74S373)[1] - TIVOVmax(8286) - TDVCLmin(8086)

= 2(200) - 35 - [4(40) + 85] - 110 - 30[1] - 30 - 30

= - 80 =>1 wait state needed (N = 1)

WRITE DATA SET-UP AND HOLD TIME MARGINS

In write cycles, the write data must

1. reach the dynamic RAMs long enough before $\overline{CAS}$ to meet the RAM's data set-up time parameter, t_{DS} (Figures 32 and 33), and

2. be held long enough after $\overline{CAS}$ to meet the RAM's data hold time parameter (t_{DH}) (Figures 32 and 34.)

Data set-up time margin is calculated in Equation 5, and data hold time margin is given in Equation 6. Again, these are margins, so a positive number indicates that system timing requirements are met for worst-case timings. Data hold time is a function of the number of 8086 wait states, represented as N, as is the read data access time margin. No wait states are required to meet this parameter.

Write Data Set-Up Time Margin (Equation 5.)

= CPU $\overline{WR}$ Active to Data Valid Delay + $\overline{CAS}$ Delay - Data Delay

= TCLMLmin(8288) + t_{CC}min(8203) - TCLDVmax(8086) - TIVOVmax(8286) - t_{DS}min(2164A-20)

= 10 + [3(40) + 25] - 110 - 30 - 0

= 15

Write Data Hold Time (Equation 6.)
Margin (N = 0)

= CPU Data Hold Time, from $\overline{AMWC}$ Active + Data Delays - $\overline{CAS}$ Active Delay

= (2 + N)TCLCL(5MHz) + TCLCHmin(8284A) + TCHDXmin(8086) + TIVOVmin(8286) - TCLMLmax(8288) - t_{CC}max(8203) - t_{DH}min(2164A—20)

= 2(200) + [⅔(200) - 15] + 10 + 5 - 35 - [4(40) + 85] - 45

= 308

[1] t_p(74S373) is the greater of t_{PHL} (from data) or t_{PLH} (from data) and is compensated for V_{cc} and temperature variations, and is derated for a 300$_p$F load (T.I. spec is at 15$_p$F).

t_p(74S373) = 13ns + 0.05ns/$_p$F(300 - 15)$_p$F + 2.75ns = 30ns.

Where 13ns is T.I. spec value
0.05ns/$_p$F is derating factor for excess capacitive load
(300 - 15) is excess capacitive load
2.75 is compensation for T_A and V_{cc} variation

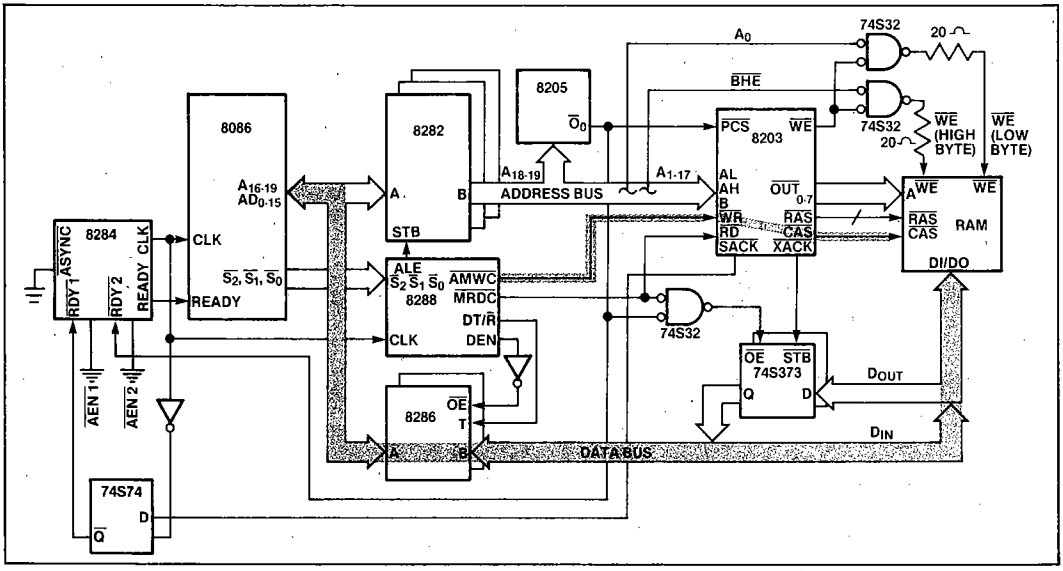


Figure 32. Write Data Set-Up and Hold Time Margins

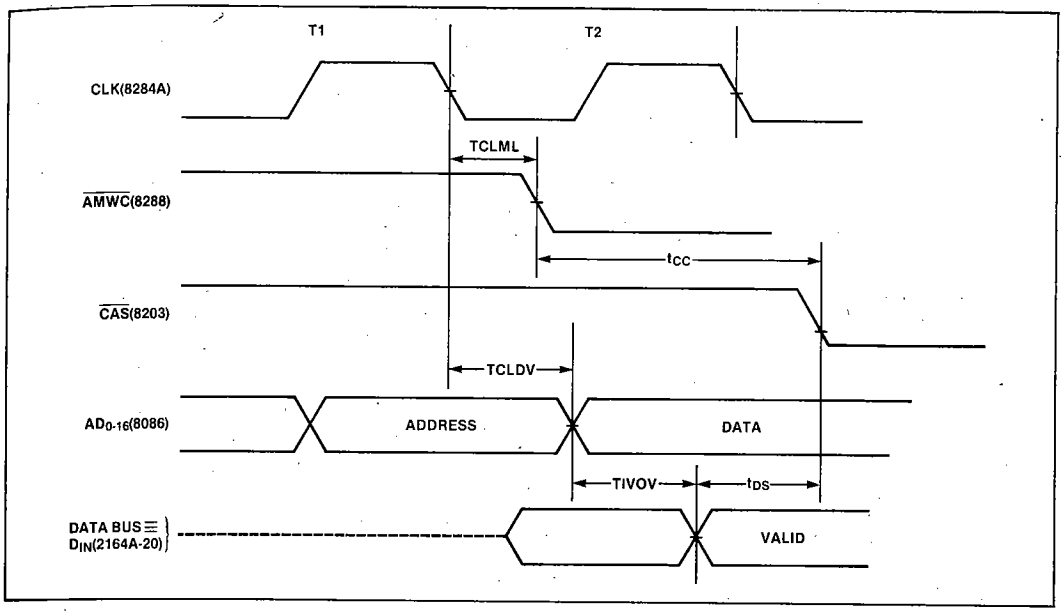


Figure 33. Write Data Set-Up Time Margin

SACK SET-UP TIME MARGIN

As explained earlier, SACK (and XACK) are "handshaking" signals used to tell the microprocessor when it may terminate the bus cycle in progress. Thus, SACK timing determines how many wait states will be generated, as opposed to how many wait states are actually required for proper operation, which is determined by the read data access time for read cycles and by the write data hold time for write cycles. If SACK causes more wait states than are required, there is a performance penalty, but the system operates; if too few wait states are generated, the system will not function.

SACK and XACK serve the same function; they differ only in timing. XACK is Multibus compatible, and is activated only when the read data is actually on the bus (in a read cycle) or when the write data has been latched into the RAM (in a write cycle). SACK is activated earlier in the memory cycle than XACK to compensate for delays in the microprocessor responding to this signal to terminate the cycle. Use of SACK is normally preferable, as it results in the fewest possible wait states being generated. But in some systems, SACK will not generate a sufficient number of wait states, so XACK or a delayed form of SACK must be used. Note that the number of wait states generated by SACK and XACK will vary, depending on whether a refresh cycle is in progress when the memory cycle was requested, and if refresh cycle is in progress, how near it is to completion. SACK is sampled by the 8284A Clock Generator Chip's RDY1 or RDY2 input. The 8284A can be programmed to treat these inputs as either synchronous or asynchronous inputs by tying its ASYNC input (pin 15) either high or low, respectively. SACK must be treated as asynchronous unless it has been synchronized to the microprocessor's clock with an external flip-flop.

SACK set-up time is shown in Figures 35 and 36, and is calculated in Equation 7. This equation indicates that, at worst case, one wait state will be generated (n = 1). This satisfies the requirements of the system, namely one wait state for reads and zero (or more) wait states for writes.

SACK Set-Up Time Margin (N = 0) (Equation 7.)

= RD or WR Active to SACK Active Delay

= (N)TCLCL(5MHz) + t_{PLH}min(7404)[1] - TCLMLmax(8288) - t_{CA}max(8203) - t_{SU}min(74S74)

= 0 + 1 - 35 - [2(40) + 47] - 3

= -164 ⇒ 1 wait state wil be generated (N = 1)

We have only looked at "worst case" SACK set-up time so far, to determine the maximum number of wait states that will be generated (assuming no delays due to a refresh cycle in progress). We should look at "best

[1] Not specified — use 1 ns.

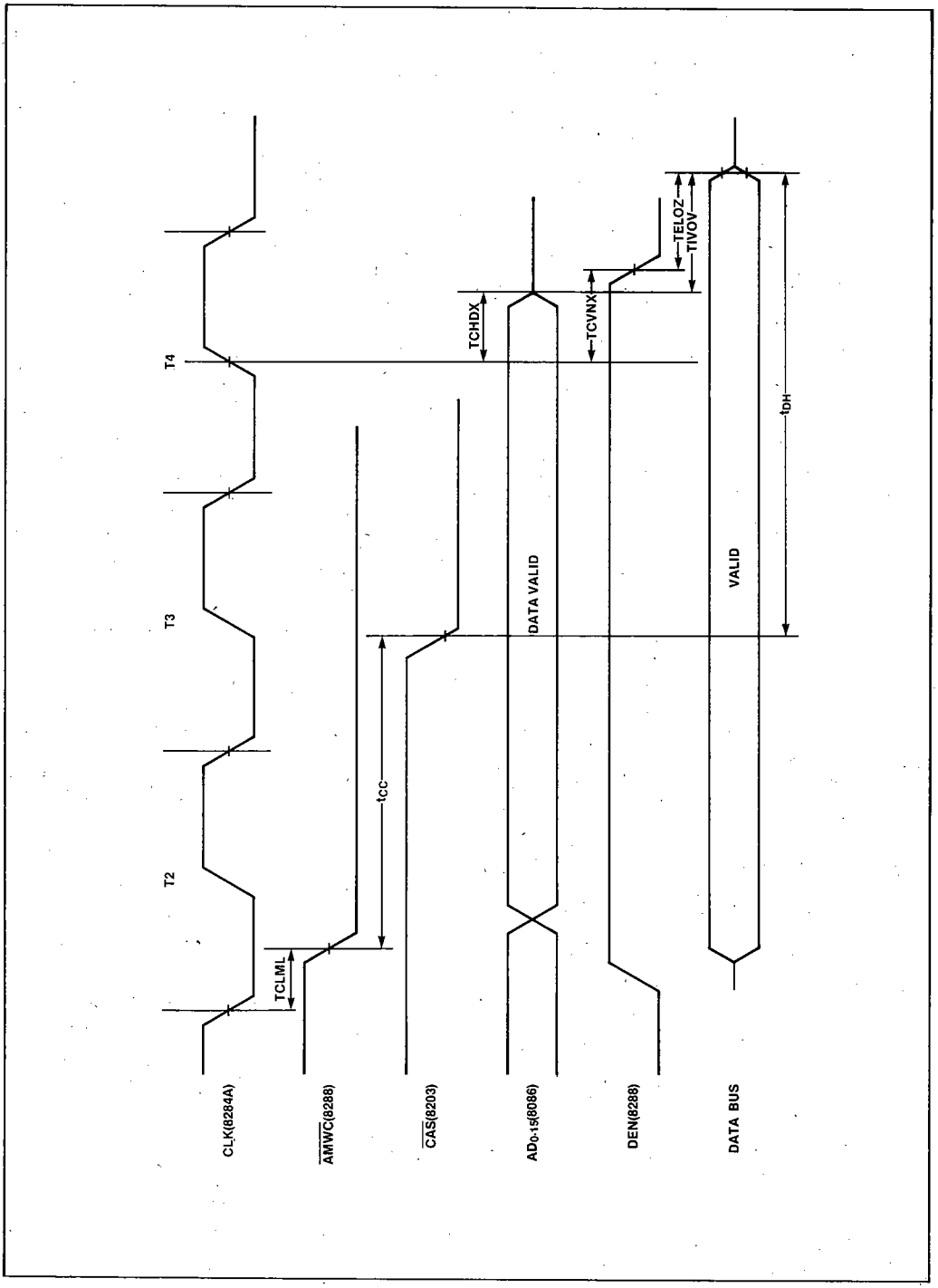


Figure 34. Write Data Hold Time Margin

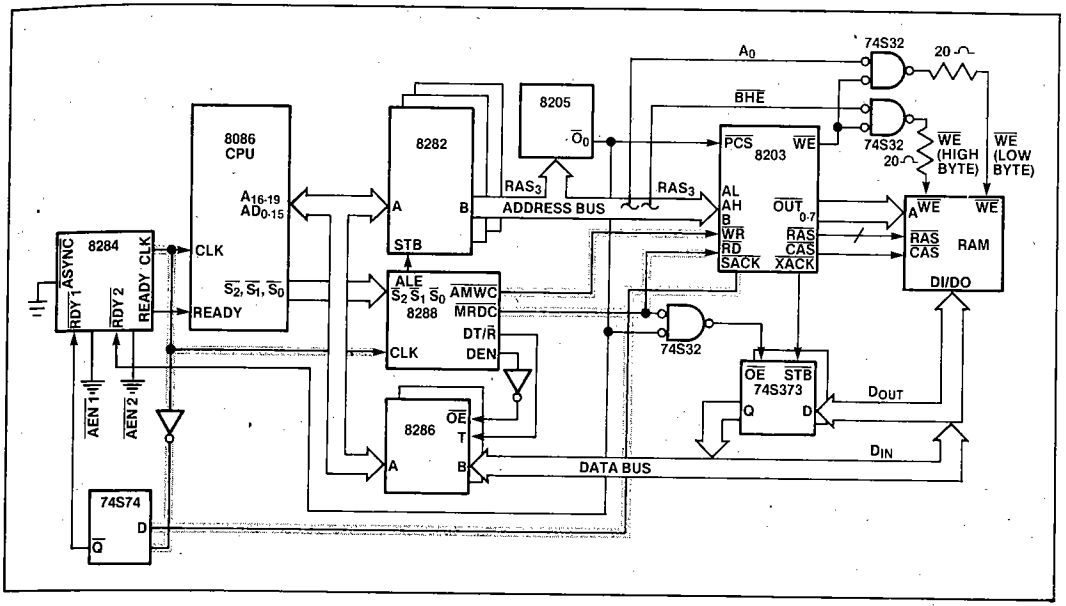


Figure 35. SACK Set-Up Time Margin

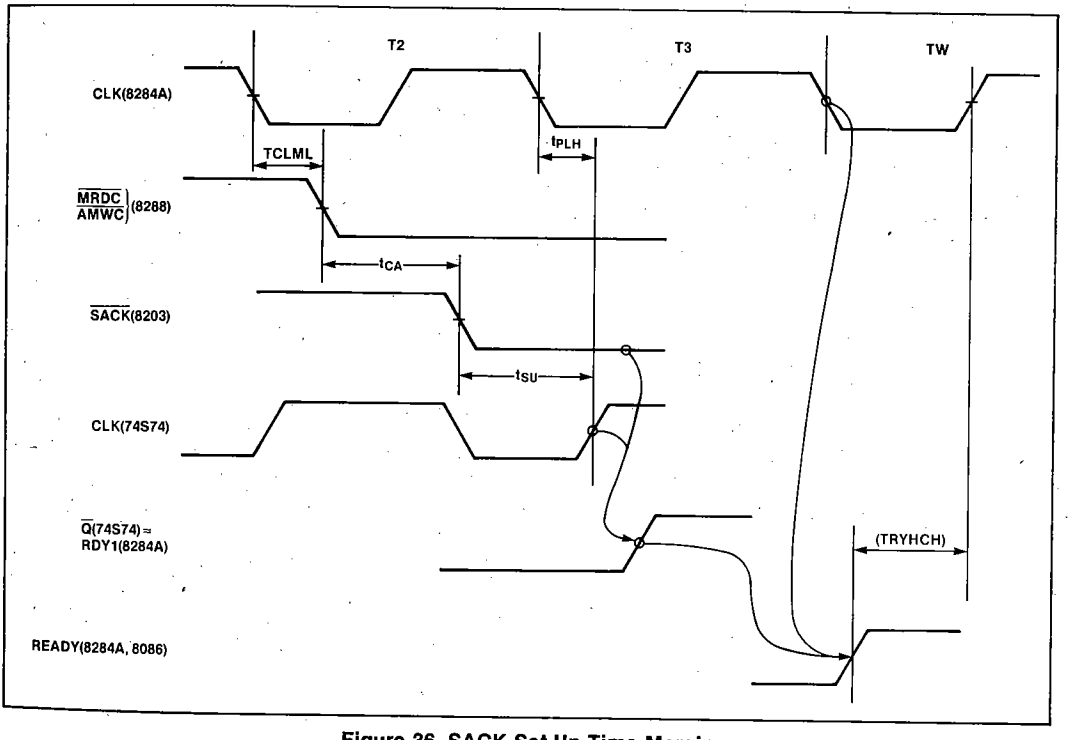


Figure 36. SACK Set-Up Time Margin

case" $\overline{SACK}$ timing also, to make sure enough wait states are always generated. Note that in Figure 35, $\overline{SACK}$ goes through an external 74S74 flip-flop; this samples $\overline{SACK}$ on-half clock cycle earlier than the 8284A does (on the same clock edge that activates $\overline{MRDC}$ or $\overline{AMWC}$), effectively reducing $\overline{SACK}$ set-up time by one-half clock period. This guarantees the proper number of wait state will be generated for "best case" $\overline{SACK}$ timing. Adding this flip-flop does not increase the worst case number of wait states generated by $\overline{SACK}$.

In the case where a memory cycle is requested while a refresh cycle is in progress, the memory cycle will be delayed by a variable amount of time, depending on how near the refresh cycle is to completion. This delay may be as long as one full memory cycle if the refresh was just starting; this time is about 650 ns, depending on the 8203's clock frequency. $\overline{SACK}$ set-up, read data set-up, and write data hold times to the microprocessor's clock are not the same as in the usual case where there is no refresh interference. In this case, $\overline{SACK}$ is delayed until the read or write cycle has been completed by the RAM, so that there is no possibility of terminating the cycle too soon.

PCS SET-UP TIME MARGIN

The 8203's $\overline{RD}$, $\overline{WR}$, and ALE inputs must be qualified by $\overline{PCS}$ in order to perform a memory cycle. If the $\overline{PCS}$ active set-up time parameter (t_{PCS}) is violated, the memory cycle will be delayed. In this case all maximum delays normally measured from command (t_{CR}, t_{CC}, t_{CA}) will be measured instead from $\overline{PCS}$ active and will be increased by t_{PCS} (20 ns). Minimum t_{CR}, t_{CC}, t_{CA} delays remain the same, but are measured from command or $\overline{PCS}$ whichever goes active later. If t_{PCS} is violated, care must be taken that $\overline{PCS}$ does not glitch low while $\overline{RD}$, $\overline{WR}$, or ALE is active, erroneously triggering a memory cycle. t_{PCS} is not violated in this system, however (Equation 8).

$\overline{PCS}$ Set-Up Time Margin (Equation 8.)

= CPU Address Valid to Command Active Delay - $\overline{PCS}$ Decode Time

= TCLCL(5MHz) + TCLMLmin(8288) - [Greater of TCLAVmax(8086) + TIVOVmax(8282) or TCLLHmax(8288) + TSHOVmax(8282)] - $t_p$max(8205) -t_{PCS}min(8203)

= 200 + 10 - [Greater of (110 + 30) or (15 + 45)] - 18 - 20

= 32

RAM DATA OUT HOLD TIME MARGIN

The 8203 CAS output is only held valid for a fixed length of time during a read cycle, after that the RAM data outputs are 3-stated. This time is not long enough to allow the 8086 to read the data from the bus, so the data must be latched externally. This latch should be a transparent type and should be strobed by $\overline{XACK}$ from the 8203. Because the minimum time from $\overline{XACK}$ active to $\overline{CAS}$ inactive is only 10 ns, a latch with a data hold time requirement of 10 ns or less (such as a 74S373) should be used (see Equation 9).

RAM Data Out Hold Time Margin, from $\overline{XACK}$ Active (Equation 9.)

= t_{ACK}min(8203) + t_{OFF}min(2164A - 20) - $t_H$min(74S373)[1].

= 10 + 0 - 10

= 0

OTHER CALCULATIONS

Equations 3, 4, 6 and 7 may be solved directly for N, where N is the number of wait states, to find how many wait states are required at a given frequency. Alternatively, a number may be substituted for N and these equations solved for the 8086's clock period, TCLCL, to find the maximum microprocessor frequency possible with N wait states. Note that the clock high and low times (TCHCL and TCLCH) are also a function of TCLCL. Be sure to use the proper speed selection of the 8086 in this calculation, as various A.C. parameters are different and the result may be different for different speed selections of the 8086, even at the same frequency. Be sure to check the other equations at this frequency to make sure they are OK, too.

Finally, for given values of TCLCL and N, Equations 3, 4, 6, and 7 may be checked to find the lowest 8203 clock frequency which will allow the same system performance, if it is desired to operate at some frequency other than the 25 MHz we assumed.

CONCLUSION

This design will operate with, at worst case, one wait state (except for refresh) at microprocessor frequencies up to 6 MHz, using slow (200 ns access time) dynamic RAMs. At 6 MHz, it is limited by a lack of $\overline{SACK}$ set-up

[1] A 74S373 must be used to meet this timing requirement. Even though worst case margin is 0 ns, this is not a critical timing, as valid data will hold on the latch inputs for a considerable time after the RAM outputs 3-state.

time. At 5 MHz, the 8203 can be operated at any clock frequency from 18.432 MHz to 25 MHz, still with only one wait state.

Example 2. 8086 Alternate Configuration System (5 MHz)

Figure 37 shows another 8086 Max mode system at 5 MHz, but this time using the Alternate Configuration, which allows it to operate with *no wait states* (except for refresh).

The system in the previous example was limited by $\overline{SACK}$ set-up time. $\overline{SACK}$ set-up time can be improved by sampling $\overline{SACK}$ later; this has been done by changing the clock edge used to sample $\overline{SACK}$, allowing roughly ⅓ clock period longer. $\overline{SACK}$ set-up time (and read data access time and write data hold time) margin can also be improved by activating the $\overline{RD}$ or $\overline{WR}$ inputs of the 8203 earlier in the 8086's bus cycle; this is the purpose of the extra logic in Figure 37 (I.C.s A8 - A11). These generate advanced $\overline{RD}$ and $\overline{WR}$ signals timed from the falling edge of ALE, which occurs roughly ⅓ clock period sooner than the $\overline{MRDC}$ and $\overline{AMWC}$ are generated by the 8288 Bus Controller. Altogether, these changes allow about one 8086 clock period more set-up time for $\overline{SACK}$.

Let's look at this logic in more detail. An Intel 8205 (A8) is used to decode the 8086's status outputs $\overline{S_{0\text{-}2}}$. An opcode fetch, memory read, or memory write decode to 8205 outputs 4, 5, and 6, respectively. These outputs go to the D inputs of two 74S74 flip-flops. The Q output of flip-flop A10.2 is an advanced memory read signal and the Q output of A11.2 is an advanced memory write signal. As shown in Figure 37, the 8203 is not activated for opcode fetches, but it can be if 8205 outputs 4 and 5 are ORed with the unused 74S00 gate (A9.4) and the $\overline{Q}$ output of A10.2 used instead of Q. Both flip-flops are clocked by the falling edge of ALE to generate the advanced commands. Flip-flop A10.1 is clocked by the trailing edge of either $\overline{AMWC}$ (Advanced Memory Write Command) or $\overline{MRDC}$ (Memory Read Command) from the 8288 bus controller (A6), indicating that the 8086 has completed the memory cycle. A10.1, in turn, presets both the A10.2 and A11.2 flip-flops to terminate the advanced memory read and write signals to the 8202A. A10.1 is then preset to its initial state by ALE going active at the start of the next bus cycle.

Because RAM write cycles are started very early in the 8086's bus cycle using this logic, the 8203 will activate $\overline{CAS}$ to the RAMs (latching write data) before the data is valid from the 8086. This requires delaying $\overline{WE}$ to the RAMs and performing a "late write" (explained earlier under *Dynamic RAMs*) in order to allow more time for the write data to arrive. But the $\overline{WE}$ signal must not be delayed so long that there is no longer enough data hold time, measured from when $\overline{WE}$ goes active; or that the $\overline{WE}$ active to $\overline{CAS}$ inactive delay spec or the RAM (t_{RWL}) is violated. None of the control signals from the 8086 or 8288 bus controller satisfy both of these timing constraints, so such a signal is generated by flip-flop A11.1, which serves to delay $\overline{AMWC}$ from the bus controller by an amount of time equal to TCLCH (the low time of the 8086's clock). A11.1 is also preset by A10.1 at the end of the memory cycle. The Q output of A11.1 is ANDed with $\overline{WE}$ from the 8203 by A14.1 to form a delayed RAM $\overline{WE}$. As in the previous example, this signal is then ANDed with $\overline{BHE}$ and AO to form the $\overline{WE}$ for the high and low bytes of RAM, respectively.

A total of four packages (three 14-pin and one 16-pin) of TTL logic are required.

The dynamic RAM interface timings are identical to the last example (Equations 1 (a)-(h)); 2164A-20 RAMs will be used again.

ADDRESS SET-UP AND HOLD TIME MARGINS

Address set-up and hold time margins are given in Equations 10 and 11, respectively. An 8086-*2* microprocessor has been used instead of the standard 8086, as this speed-selected part gives better address set-up to $\overline{RD}$ or $\overline{WR}$ times, which this design needs since it uses advanced $\overline{RD}$ and $\overline{WR}$ commands.

Row Address Set-Up Time Margin[1] (Equation 10.)

= CPU Address to Adv. $\overline{RD}$ Delay
+ $\overline{RAS}$ Delay - Address Delays

= TCLCHmin(8284A) + TCHLLmin(8288)[2]
+ t_{PLH}min(74S00)[3] + t_{PHL}min(74S74)[2]
+ t_{CR}min(8203) - [Greater of
TCLAVmax(8086 - 2) + TIVOVmax(8282)
or TCLLHmax(8288) + TSHOVmax(8282)]
- t_{AD}max(8203) - t_{ASR}min(2164A-20)

= [⅔(200) - 15] + 2 + 1 + 2 + [(40) + 30]
- [Greater of (60 + 30) or (15 + 45)] - 40 - 0

= 63

[1] Read or write cycles only. Eq. 1b gives this timing for refresh cycles.

[2] Not specified — use 2 ns.

[3] Not specified — use 1 ns.

Address Hold Time Margin (N = 0) (Equation 11.)

= CPU Address Hold Time from Adv. $\overline{RD}$ Active - $\overline{RAS}$ Inactive Delays

= (3+N)TCLCL(5MHz) + TCHCLmin(8284A) + TCLLHmin (8288) + TSHOVmin(8282) - TCLMLmax(8288) - t_{CC}max(8203) - t_{RSH}max(8203)

= (3)200 + [⅓(200) + 2] + 2 + 5 - 35 - [4(40) + 85] - [5(40) + 20]

= 175

READ DATA ACCESS TIME MARGIN

Read data access time margin is shown in Equation 12; no wait states are required for read cycles, even with 200 ns access time RAMs.

Read Data Access Time Margin (N = 0) (Equation 12.)

= Adv. $\overline{RD}$ to Data Valid Delay - $\overline{CAS}$ Delay - Read Data Delays

= (2+N)TCLCL(5MHz) + TCHCLmin(8284A) - TCHLLmax(8288) - t_{PLH}max(74S00) - t_{PHL}max(74S74) - t_{CC}max(8203) - t_{CAC}max(2164A—20) - t_{p}max(74S373) - TIVOVmax(8286) - TDVCLmin(8086—2)

= (2)200 + [⅓(200) + 2] - 15 - 5 - 10 - [4(40) + 85] - 110 - 30 - 30 - 20

= 3

WRITE DATA SET-UP AND HOLD TIME MARGINS

Write data set-up and hold times are shown in Equations 13 and 14, respectively. No wait states are required during write cycles. Note that write data set-up has been guaranteed by delaying $\overline{WE}$ from the 8203 with clocked $\overline{AMWC}$ from the bus controller and performing "late write" cycles; write data set-up time would not be satisfied otherwise. Equation 15 verifies that $\overline{WE}$ has not been delayed too long to meet the RAM's $\overline{WE}$ active to $\overline{RAS}$ inactive set-up time (t_{RWL}). The RAM's $\overline{WE}$ active to $\overline{CAS}$ inactive set-up time (t_{CWL}) is also satisfied, since $\overline{CAS}$ does not go inactive until at least 20 ns after $\overline{RAS}$.

Write Data Set-Up Time Margin (Equation 13.)

= CPU Data to Clocked $\overline{AMWC}$ Set-Up + $\overline{WE}$ Delays - Data Delays

= TCLCHmin(8284A) + t_{PHL}min(74S74)[1] + (2)t_{PHL}min(74S32)[1] - TCLDVmax(8086-2) - TIVOVmax(8286) - t_{DS}min(2164A—20)

= [⅔(200) - 15] + 2 + (2)2 - 60 - 30 - 0

= 34

Write Data Hold Time Margin (N = 0) (Equation 14.)

= CPU Data Hold Time from Clocked $\overline{AMWC}$ + Data Delays - $\overline{WE}$ Delays

= (2 + N)TCLCL(5MHz)

= TCHDXmin(8086—2) + TIVOVmin(8286) - - t_{PHL}max(74S74) - (2)t_{PHL}max(74S32) - t_{DH}min(2164A—20)

= (2)200 + 10 + 5 - 10 - (2)7 - 45

= 346

$\overline{WE}$ Active Set-Up Time Margin to $\overline{RAS}$ Inactive (Equation 15.)

= TCHLLmin(8284A)[1] + t_{PLH}min(74S00)[2] + t_{CC}min(8203) + t_{RSH}min(8203) - t_{SKEW}(74S74)[3] -(2)t_{PHL}max(74S32) - t_{RWL}min(2164A—20) - TCLCL(5MHz)

= 2 + 1 + [3(40) + 25] + [5(40) - 30] - 2 - (2)7 - 50 - 200

= 52

$\overline{SACK}$ SET-UP TIME MARGIN

Equation 16 shows that $\overline{SACK}$ set-up time is satisfied; no wait states will be generated for read or write cycles (except for refresh).

$\overline{SACK}$ Set-Up Time Margin (N = 0) (Equation 16.)

= (1 + N)TCLCL(5MHz) - TCHLLmax(8288) - t_{PLH}max(74S00) - t_{PHL}max(74S74) - t_{CA}max(8203) - t_{SU}min(74S74)

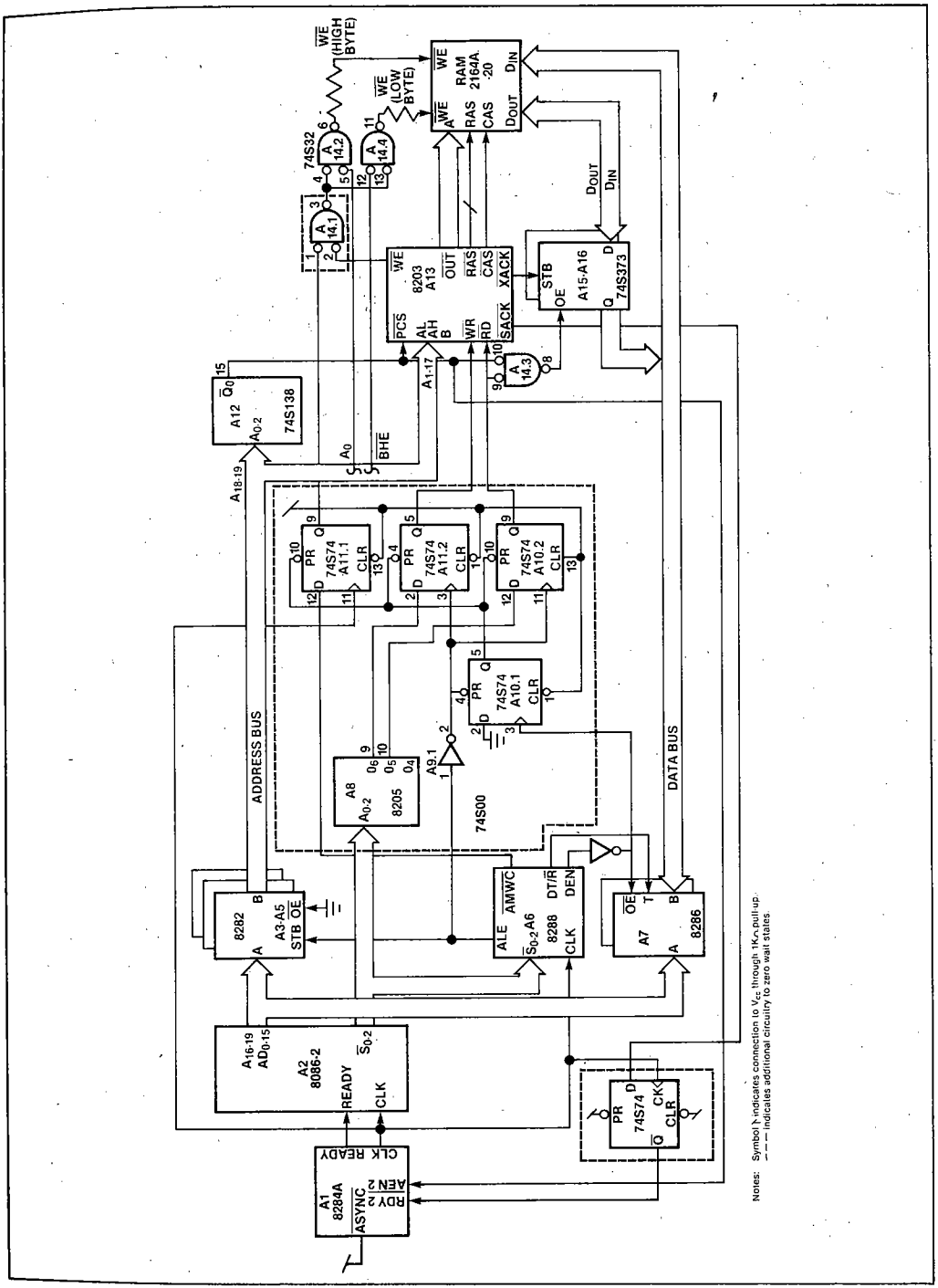
= 200 - 35 - 5 - 10 [2(40) + 47] - 3

= 20

[1] Not specified — use 2 ns.

[2] Not specified — use 1 ns.

[3] t_{SKEW}(74S74) is max. skew between t_{PHL}(Q output, from CLK) of two Q outputs in same package — use = 2 ns.



Notes: Symbol ⟩ indicates connection to V_{CC} through 1KΩ pull up.
– – – indicates additional circuitry to zero wait states.

Figure 37. 8086 Alternate Configuration System

$\overline{PCS}$ Set-Up Time Margin (Equation 17.)

= CPU Address Valid to Adv. $\overline{RD}$ or Adv. $\overline{WR}$ Delay - $\overline{PCS}$ Decode Time

= TCLCHmin(8284A) + TCHLLmin(8288)[1] + t_{PLH}min(74S00) + t_{PHL}min(74S74)[1] - TCLAVmax(8086—2) - TIVOVmax(8282) - $t_p$max(74S138[3] - t_{PCS}min(8203)

= [⅔(200) - 15] + 2 + 1 + 2 - 60 - 30 - 12 - 20

= 1

PCS SET-UP TIME MARGIN

$\overline{PCS}$ set-up time for the 8203 (t_{PCS}) is satisfied, but not with as much margin in the last example (Figure 17). This is because the $\overline{RD}$ and $\overline{WR}$ commands are activated earlier in the microprocessor's bus cycle, leaving less time to decode $\overline{PCS}$ from the address bus.

[1] Not specified — use 2 ns.
[2] Not specified — use 1 ns.
[3] Must use 74S138 to maintain $\overline{PCS}$ Set-Up Time Margin.

CONCLUSION

This design will operate with a guaranteed zero wait states up to 5 MHz using slow (200 ns access time) RAMs. At this frequency, it is limited by both read and write data set-up times, and to a lesser extent, by $\overline{SACK}$ set-up time. Using faster RAMs will not raise the maximum frequency, as write data and $\overline{SACK}$ set-up times are not affected by the RAM speed. The 8203 operating frequency must be 25 MHz.

This design can be used (with some modifications) to allow one wait state performance up to 8086 clock frequency of 8 MHz.